TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ERKKI CASTRÉN

OPENCRP – AVOIMEN LÄHDEKOODIN PILVIPALVELU- JA MOBIILIROBOTTIYMPÄRISTÖ

Diplomityö

Tarkastajat: professori Pekka Loula
ja TkT Petri Oksa
Tarkastajat ja aihe hyväksytty
Talouden ja rakentamisen tiedekuntaneuvoston kokouksessa 8. kesäkuuta 2016

TIIVISTELMÄ

ERKKI CASTRÉN: OpenCRP – Avoimen lähdekoodin pilvipalvelu- ja mobiilirobottiympäristö
Tampereen teknillinen yliopisto, Porin laitos
Diplomityö, 67 sivua, 12 liitesivua
Lokakuu 2016
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotekniikka
Tarkastajat: professori Pekka Loula ja TkT Petri Oksa

Avainsanat: OpenCRP, pilvipalvelu, mobiilirobotti, avoin lähdekoodi, ROS, Hadoop

Tampereen teknillisen yliopiston Open Cloud Robotic Platform -projektin (OpenCRP) tarkoituksena on muun muassa tiedon tuottaminen ja osaamisen kartuttaminen avoimen lähdekoodin mobiilirobottien ja -pilvipalvelun hyödyntämisestä monirobottiympäristössä. Projektiin kuuluu demonstraatio- ja pilotointiympäristön rakentaminen ja sen ominaisuuksien tutkiminen. Projektin kuluessa tulimme vakuuttuneiksi siitä, että monirobottiympäristön toteuttaminen kannattaa lykätä siihen asti, kunnes sen edistyneemmin toteuttavasta ROS 2:sta on käytettävissä valmiimpi versio. Siksi tämä työ kattaa ympäristön rakentamisen vain yhdelle mobiilirobotille, sen ohjaukseen tarkoitetulle käyttöliittymälle ja karttojen tallennuspaikkana toimivalle pilvipalvelulle. Lisäksi työssä tarkastellaan valmiin ympäristön ominaisuuksia.

Robottialustana käytettiin kahta Turtlebot II:ta, toinen varustettuna Kinect Xbox 360- ja toinen Xtion Pro Live -3D-sensorilla. Turtlebotien ajamista varten niiden päälle valmisteltiin kannettavat tietokoneet, joihin asennettiin ROS Indigo- ja Hadoop 2.7.2 -ohjelmistot Ubuntu 14.04 alustalle. Robotin etäohjauksen mahdollistamiseksi ROS asennettiin myös kahteen muuhun tietokoneeseen. Näihin etäkoneisiin asennettiin myös Hadoop ja ne konfiguroitiin HDFS-pilvipalveluklusteriksi, jota voitiin terminaalikomennoilla käyttää karttatiedostojen tallennuspaikkana. Samalla niihin konfiguroitiin valmiiksi MapReduce mahdollista tulevaisuuden tarvetta ajatellen. Robotin ohjausta varten toteutettiin Qt:llä yksinkertainen käyttöliittymä. Sen avulla robotti voidaan, joko sen omalta koneelta tai etäkoneelta, ohjata kulkemaan haluttuihin karttapisteisiin. Robotti yhdistettiin muihin koneisiin WLANilla, ja sen kartoitus- ja navigointikykyä testattiin. Molempien robottien 3D-sensorien ominaisuuksia testattiin ja WLAN-yhteyden sopivuutta ympäristöön arvioitiin.

Lopputuloksena oli toimiva avoimen lähdekoodin mobiilirobottiympäristö, jossa robotin kartat voitiin tallentaa ja niitä noutaa Hadoop-pilvipalveluklusterista. Robottia voitiin ohjata toteutetun käyttöliittymän avulla ja matkallaan se osasi väistää eteen tulevia esteitä. Saimme paljon tietoa OpenCRP-ympäristön käytännön toteuttamisesta ja ympäristöön liittyvistä haasteista. Ympäristön jatkokehitystä ajatellen esiin nousi useita toimenpiteitä, joilla sen toimintaa voidaan parantaa. Tärkeimpinä voidaan mainita robotin lähihavainnoinnin ja verkkoyhteyden parantaminen.

ABSTRACT

ERKKI CASTRÉN: OpenCRP – Open Source Cloud Service and Mobile Robot Environment
Tampere University of Technology, Pori Department
Master of Science Thesis, 67 pages, 12 Appendix pages
October 2016
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiners: Professor Pekka Loula and D.Sc. (Tech.) Petri Oksa

Keywords: OpenCRP, Cloud Service, Mobile Robot, Open Source, ROS, Hadoop

The Open Cloud Robotic Platform project (OpenCRP) at the Tampere University of Technology aims to gather information and increase the knowledge on using open source mobile robots and cloud services in a multirobot ecosystem. The project includes implementation of a demonstration/pilot environment as well as its examination. When proceeding deeper into the research we found that ROS 2 (Robot Operating System 2) will have simpler implementation of the multirobot feature than ROS 1 has. We became convinced that it is worth waiting for a more mature ROS 2 release and implement the multirobot environment when the upgrade to ROS 2 becomes worthwhile. Thus, this thesis covers only an implementation of a single robot environment with a controlling user interface and a cloud service as a map storage. Features of the environment are examined.

We had two Turtlebot II robot development kits available. Both of the robots were equipped with 3D sensors, one with a Kinect Xbox 360 and the other with an Xtion Pro Live. To drive the robots, two laptop computers were prepared by installing ROS Indigo and Hadoop 2.7.2 software on their Ubuntu 14.04 platforms. Laptops were placed on top of the robots and attached to them via USB connections. To control a robot remotely, ROS was installed into two other computers as well. Also, Hadoop was installed into these remote computers and they were configured as a two computer HDFS-cluster. The cluster served as a map storage and could be used from the robots with terminal commands. Remote computers were also configured to have MapReduce available for possible usage in the future. For robot controlling purposes a simple UI was implemented with Qt. With the UI in one or more of the computers, a robot can be driven to desired map positions by press of a button. A WLAN was used to connect the robot's laptop to the remote computers and its mapping and navigation capabilities were tested. Both 3D sensors' observation ranges were measured and the adequacy of a WLAN connection of robot's computer was evaluated.

The result we achieved was a fully functional open source mobile robot environment, where the maps could be stored into and fetched from a Hadoop cloud service. A robot could be controlled via implemented UI avoiding obstacles on its path. We gained a lot of knowledge about implementing an OpenCRP environment in practise and the challenges of the implemented environment. Many possibilities for improving the functionality of the environment arose. The two most important would be improving the near range obstacle observation capabilities and the quality of wireless connection.

ALKUSANAT

Etsiessäni sopivaa aihetta diplomityötä varten, sain tilaisuuden osallistua Tampereen teknillisen yliopiston Porin laitoksen OpenCRP-projektiin. Tämä diplomityö perustuu projektissa kertyneeseen aineistoon.

Haluan kiittää työni tarkastajia professori Pekka Loulaa ja projektipäällikkö Petri Oksaa kannustuksesta ja taustatuesta diplomityöni kirjoittamisen aikana ja erityisesti mielenkiintoisen työpaikan tarjoamisesta projektiryhmässä. Lisäksi kiitän laboratorioinsinööri Teemu Alapaholuomaa, lehtori Juha Vihervaaraa, yliopisto-opettaja Matti Monnosta ja lehtori Frank Cameronia saamistani neuvoista. Kiitokset myös muulle laitoksen henkilökunnalle, joka oli aina valmis auttamaan.

Porissa, 15.10.2016

Erkki Castrén

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

ACL	Access Control List, pääsynhallintalista
AMCL	Adaptive Monte Carlo Localization, todennäköisyysperusteinen 2-D paikannusmenetelmä
API	Application Programming Interface, ohjelmointirajapinta
BSD	Berkeley Software Distribution, open source-lisenssi
CI	Continuous Integration, jatkuva integraatio
DAvinCi	Distributed Agents with Collective Intelligence, pilvipalvelurobottiympäristö
DDS	Data Distribution Service, tiedonvälitystä reaaliaikajärjestelmissä tarjoava väliohjelmisto tai sen määrittävä standardi
Distro	Distribution, jakeluversio
FPS	Frames Per Second, kuvataajuus
Git	Linus Torvaldsin alulle laittama ohjelmistojen versionhallintatyökalu
GMapping	Grid Mapping, paljon käytetty SLAM-algoritmi
HDFS	Hadoop Distributed File System, Hadoopin hajautettu tiedostojärjestelmä
HTTP	Hypertext Transfer Protocol, sovellusprotokolla hypertekstі-informaatiojärjestelmille
IDE	Integrated Development Environment, integroitu ohjelmointiympäristö
IFR	International Federation of Robotics, Kansainvälinen robotiikan keskusjärjestö
IR	Infra Red, infrapuna
LTS	Long Term Support, ohjelmaversiota tuetaan määrätty pidempi aika, yleensä 3-5 vuotta
MD5	Message Digest 5, algoritmi tiivisteen laskemiseksi datasta
MIMO	Multiple-Input and Multiple-Output, 802.11n -standardin sisältämä, useaa antennia käyttävä ja monitie-etenemiseen perustuva tiedonsiirtotekniikka
MU-MIMO	Multi User MIMO, 802.11 ac -standardin sisältämä monta yhtaikaista käyttäjää (datavirtaa) mahdollistava MIMO-tekniikka
OpenCRP	Open Cloud Robotic Platform, avoin pilvipalvelurobottialusta
open source	avoin lähdekoodi
P2P	Peer to Peer, vertaisyhteys, vertaisverkko
PGM	Portable Graymap format, harmaasävy-bitmap formaatti
PPA	Personal Package Archive, henkilökohtainen pakettivarasto, tapa jakaa Debian-paketteja
QoS	Quality of Service, tietoliikenteen luokittelu ja priorisointi
RGB	Red Green Blue, värimalli, jossa värit muodostetaan sekoittamalla punaista, vihreää ja sinistä
ROS	Robot Operating System, avoimen lähdekoodin robottikäyttöjärjestelmä
ROSCon	Annual ROS developer’s conference, vuotuinen ROS-kehittäjien konferenssi
roscpp	ROSin C++ toteutus, tarjoaa C++ client-kirjaston

RPC	Remote Procedure Call, etäproseduurikutsu. RPC-protokollan avulla voidaan kutsua toisissa prosesseissa (myös toisissa koneissa ajettavissa) sijaitsevia proseduureja.
RSA	julkisen avaimen salausalgoritmi, jonka nimi muodostuu kehittäjien (Rivest, Shamir, Adleman) sukunimien alkukirjaimista
Rsync	Remote Sync, tiedostojen etäsynkronointi-ohjelma
RTPS	Real-Time Publish-Subscribe, IP-pinon päällä toimiva tiedonsiirtoprotokolla
SLAM	Simultaneous Localization and Mapping, robottikartoituksessa yhtaikaisen kartoituksen ja paikannuksen ongelma
SSH	Secure Shell, tietoliikenteen salaukseen käytetty protokolla
SU-MIMO	Single User MIMO, yhtä käyttäjää kerrallaan palveleva MIMO-tekniikka
Turtlebot II	robottialusta harrastus-, tutkimus- ja opetuskäyttöön
UI	User Interface, käyttöliittymä
WDS	Wireless Distribution System, tekniikka, jolla WLAN-verkkoa voidaan laajentaa liittämällä siihen langattomasti lisää yhteyspisteitä
XML	eXtensible Markup Language – koodausjärjestelmä, joka mahdollistaa dokumentin lukemisen sekä ihmiselle että koneelle
XMLRPC	XML-Remote Procedure Call, XML-etäproseduurikutsu
YAML	YAML Ain’t Markup Language, XML:ää yksinkertaisempi rakenteinen formaatti
YARN	Yet Another Resource Negotiator, MapReducen versio 2

.

1. JOHDANTO

Robotiikka on maailmalla ollut jo pitkään kasvavan kiinnostuksen- ja kehityksen kohteena. Jatkuvasti etsitään uusia tapoja hyödyntää robotiikkaa muuallakin, kuin puhtaasti teollisessa tuotannossa. Robotit yleistyvätkin kasvavalla vauhdilla kotitalouksissa.

IFR:n (International Federation of Robotics) arvioiden mukaan kotitalouksien palvelurobottien myynti, v. 2014 oli n. 3.3 miljoonaa kappaletta. Vuosien 2015-18 välisenä aikana IFR arvioi kotitalouskäyttöön myytävän jopa 25.9 milj. palvelurobottia. Näihin lukuihin on laskettu ruohonleikkuu-, pölynimuri-, ikkunanpesu-, uima-altaan puhdistus- ja muut sen tapaiset kotitalouksien hyötyrobotit, ei viihdekäyttöön tai leluksi tarkoitettuja laitteita. [1]

1.1 Pilvipalvelut robotiikassa

Pilvipalvelujen liittäminen robottiympäristöön antaa mahdollisuuden siirtää esimerkiksi vaativaa laskentaa pilvessä suoritettavaksi ja näin pienentää robotilta itseltään vaadittavaa laskentatehoa. Pilvessä laskenta voidaan hajauttaa useille koneille, jolloin perusyksikkönä käytettävä kone voi olla ominaisuuksiltaan tavallinen työasema vaativankin laskennan tapauksessa.

Pilvipalveluun voidaan myös tallentaa robottien tarvitsemaa dataa, kuten esimerkiksi toiminta-alueiden karttoja. Tällöin jokaisen samalla alueella toimivan robotin ei tarvitse erikseen kartoittaa aluetta, vaan riittää, kun yksi robotti siirtää muodostamansa kartan muiden robottien saataville pilveen. Edellytyksenä on yhteinen tietoformaatti tai tietoformaatin muunnos tarvittaessa kunkin robotin vaatimusten mukaan.

Tiedon hallittu hajautettu käsittely ja säilytys tuovat mukanaan myös vikasietoisuuden. Laskentaan käytetyn koneen vikaantuessa muut laskentaverkon koneet voivat korvata sen. Dataa taas voidaan säilyttää redundantisti useammalla palvelimella, jolloin vikatilanteiden varalle voi olla saatavilla useampiakin varmuuskopioita.

Pilvipalvelun ja robottien yhdistämisestä on maailmalla tätä kirjoitettaessa tehty jo useampia projekteja, esimerkkeinä voidaan mainita RoboEarth-projekti ja DAvinCi-ympäristö. [2], [3]

EU-rahoitteisessa RoboEarth-projektissa vv. 2010-14 yhdistyi tiedon tallentaminen ja jakaminen robottien yhteisesti ymmärtämässä formaatissa, vaativan laskennan siirtäminen robotista pilveen ja robottien välinen yhteistyö pilvipalvelua hyväksi käyttäen. Projektissa

syntyi RoboEarth-kieli, WWW-tyyppinen RoboEarth-tietokanta ja RoboEarth-Cloud Engine (Rapuyta). Kehittämässään ympäristössä he onnistuivat yhdistämään useamman robotin tekemiä 3-D karttoja uudeksi kartaksi. Projekti sai jatkoa muun muassa tietämyksenkäsittelyyn orientoituneesta KnowRob -projektista [4] ja pilveen liitettyihin monirobottijärjestelmiin turvallisuus- ja tarkastuskäytössä keskittyvästä spin-off -yhtiö Rapuyta-Roboticsista [5]. [2]

Singaporen Data Storage Instituten tutkijaryhmän DAvinCi (Distributed Agents with Collective Intelligence) esiteltiin jo 2010 IEEE:n robotiikka- ja automaatiokonferenssissa Alaskassa. DAvinCi on ROSia (Robot Operating System), sekä Hadoopin MapReducea ja HDFS:ää (Hadoop Distributed File System) hyödyntävä robottiekosysteemi. Nämä yhdistetään DAvinCi-palvelimella toisiinsa. Ryhmän testissä FastSLAM-algoritmin (Simultaneous Localization and Mapping) laskenta hajautettiin MapReducen avulla ja osoitettiin sen edut suurten alueiden kartoittamisessa jo pienellä 8 palvelimen Hadoop-klusterilla. Testissä käytettiin todellisen robottiympäristön sijaan valmiiksi tallennettua sensoridataa. [3]

DAvinCissa toisiinsa linkitettyjen robottien anturitiedot voidaan siirtää keskusohjaimelle globaalin kartan muodostamista varten. Tässä ratkaisussa sensoridataa voidaan kerätä myös roboteilta, joilla ei ole kykyä ajaa ROS-nodeja. Näitä varten DAvinCi-palvelin ajaa proxy ROS-nodeja, jotka keräävät näiden sensoritietoa tai lähettävät niille ohjauskomentoja esimerkiksi WLANin kautta. Karttaa tai sen osia voidaan jakaa roboteille HDFS:stä tarpeen mukaan. [3]

Näkymät ovat siis mielenkiintoiset varsinkin monirobottiympäristössä. Jokainen voi kuvitella, millaisia mahdollisuuksia avaa kyky käsitellä robottiparven sensoritietoja kokonaisuutena, kuitenkin tehokkuuden ja vikasietoisuuden nimissä hallitusti hajauttaen.

1.2 Palvelurobotit

Robotit voidaan jakaa teollisiin- ja palvelurobotteihin. Jako näiden välillä määräytyy tarkoitetun käytön perusteella. Palvelurobotti on ISO 8373:2012 standardin [6] mukaan robotti, joka suorittaa hyödyllisiä tehtäviä ihmisen tai laitteen puolesta, pois lukien teolliset automaatiosovellukset.

Palvelurobotit voidaan lisäksi jakaa ammatti- ja henkilökohtaisessa käytössä oleviin laitteisiin. Ammattikäyttöistä laitetta käytetään kaupallisiin tarkoituksiin ja sitä käyttää yleensä tehtävään asianmukaisesti koulutettu operaattori. Henkilökohtaisessa käytössä robottia ei käytetä kaupallisiin tarkoituksiin ja käyttäjä on yleensä maallikko.[6]

Parhaillaan eletään murrosta, jossa teknisiin ympäristöihin suljetut ja ihmisistä eristetyt teolliset robotit saavat seurakseen itsenäisesti liikkuvia, erilaisiin toimintaympäristöihin

ja niiden muutoksiin sopeutuvia palvelurobotteja. Erilaisia yksinkertaisia palvelurobotteja on ollut jo pitkään kuluttajien ulottuvilla, pölynimureista ruohonleikkureihin. Mobiilius on tunnusomaista tällaisille roboteille, joiden on tehtävistään suoriutuakseen usein osattava liikkua itsenäisesti.

Nykyisin robotit osaavat jo mallintaa toimintaympäristöään ja suunnitella tehtäviensä vaatimat kulkureitit. Ne osaavat toimia ympäristössä tapahtuvien muutosten mukaisesti, kuten esimerkiksi väistää vastaan tulevaa ihmistä tai toista robottia. Oppiminen on tärkeässä osassa tällaisten robottien toiminnassa, eikä niitä enää ohjelmoida tarkasti erikseen kaikkia tilanteita varten. Niiden toiminta perustuu pitkälti erilaisiin algoritmeihin, jotka huolehtivat muun muassa toimintaympäristön mallin päivittämisestä, robotin paikantamisesta ja reittien suunnittelusta. [7]

Palvelurobottien kehittyessä niitä on myös testattava ja käytettävä ympäristössä, jossa kulkee jatkuvasti ihmisiä. Tällöin turvallisuudesta huolehtiminen nousee merkittäväksi tekijäksi. ISO onkin v. 2014 julkaissut turvallisuusvaatimukset hoitoroboteille [8].

Australialaisen Queensland University of Technologyn tutkijaryhmä kirjoitti omista robotiikan turvallisuuteen liittyvistä kokemuksistaan artikkelin [9], jossa he nostivat esiin useita seikkoja, joita ei heti tule ajatelleeksi. Esimerkiksi palohälytyksen sattuessa mobiilirobotti ei saa tukkia poistumisteitä, vaan sen on hälytyksen sattuessa osattava väistää sivuun ja pysähdyttävä. Tämän periaatteessa yksinkertaisen tehtävän suorittaminen kaikissa olosuhteissa saattaa kuitenkin käydä ylivoimaiseksi. Sprinklerien laukeaminen, liekit tai sankka savu saattavat sekoittaa robotin paikannuksen ja siten estää väistämisen oikeaan paikkaan. Samoin kymmeniäkin kiloja painavan robotin putoaminen portaikkoon voisi aiheuttaa vakavia vammoja siellä oleville ihmisille. Tällaiset alueet voidaan toki rajata toiminta-alueen ulkopuolelle, mutta tällöin robotin toiminta-alue saattaa pienentyä huomattavastikin, jos sen siirtyminen alueelta toiselle käy mahdottomaksi.

Ryhmä oli huolissaan siitä, että turvallisuusnäkökohdat saattavat jäädä kovin vähälle huomiolle, kun keskitytään muihin asioihin. He kaipasivatkin robotiikan tutkimusyhteisöltä yhteistyötä robotiikan riskiarvioinnin standardiproseduurin kehittämiseksi. [9]

1.3 Työn tavoitteet

Työn tavoitteena on rakentaa OpenCRP (Open Cloud Robotics Platform) demonstraatioja pilotointiympäristö ja saada tietoa sen ominaisuuksista. Samalla on tarkoitus saada tietoa ympäristön rakentamiseen liittyvistä käytännön ongelmista ja niiden ratkaisuista.

Valmiiseen ympäristöön kuuluu mobiilirobottialusta ja sen päälle sijoitettu tietokone, joka on varustettu tarvittavilla open source -ohjelmistoilla (ROS, Hadoop). Siihen sisältyy myös kahden koneen Apache HDFS -pilvipalveluklusteri, johon robotin koneesta on

mahdollisuus tallentaa karttatiedostoja ja ladata niitä sieltä. Kartoitusta varten robottia tulee voida ohjata kaukosäätimellä.

Robotille voidaan antaa yksinkertaisen käyttöliittymän avulla kohde, johon se osaa ennalta kartoitetulla alueella suunnistaa omin avuin, myös tielleen tulevat uudet esteet väistäen. Käyttöliittymää voidaan käyttää paitsi robotin päällä olevalta koneelta, myös etänä sen kanssa samaan verkkoon yhdistetyltä etäkoneelta. Näiden käyttöliittymien tulee voida olla käytettävissä samanaikaisesti.

Heikkoa langatonta verkkoa ajatellen on oltava mahdollisuus konfiguroida robotti toimimaan ilman verkkoyhteyttä. Tällöin siihen ei luonnollisestikaan saada etäyhteyttä toiselta koneelta, eikä HDFS-palvelua voida käyttää. Myös kartoituksen tulee olla mahdollista ilman verkkoyhteyttä.

1.4 Työn rakenne

Luvussa 2 kerron työssä käytetyistä avoimen lähdekoodin ohjelmistoista, joiden varaan lähes koko järjestelmä rakentuu. Samassa luvussa käyn läpi myös tärkeimmät käytetyt laitteet ja niiden ominaisuudet, lähinnä laitetoimittajien antamien tietojen perusteella.

Luku 3 käsittää OpenCRP-ympäristön rakentamisen, siinä kohdatut ongelmat ja keinot, joilla ongelmat ratkaistiin. Lukuun sisältyy myös tehdyt mittaukset ja sellaisia työn aikana tehtyjä yleisiä havaintoja, joista katson asiasta kiinnostuneelle lukijalle voivan olla hyötyä. Luvussa verrataan myös sensorien mitattuja ominaisuuksia luvussa 2 esitettyihin.

Luku 4 sisältää päätelmät edellisessä luvussa esitetyistä mittaustuloksista ja ympäristön havaituista ominaisuuksista. Luvussa esitetään myös ympäristössä operoidessa hyväksi todettuja toimintamalleja.

Lopuksi luvussa 5 tehdään loppuyhteenveto koko työstä, eli mitä saatiin aikaan ja mihin jatkossa kannattaa keskittyä. Lisäksi esitetään ympäristön toiminnan parantamiseen tähtääviä ehdotuksia ja muita työn herättämiä ajatuksia.

2. ROBOTTIYMPÄRISTÖN RAKENTAMISESSA KÄYTETYT OHJELMISTOT JA LAITTEET

Tässä luvussa esitellään tärkeimmät avoimen lähdekoodin ohjelmistot, joita OpenCRP-ympäristössä käytettiin. Näitä ovat ROS-robottikäyttöjärjestelmä, Hadoop-pilvipalvelu, Qt-kehitysympäristö ja GitLab -versionhallinta.

Luvussa tutustutaan myös ympäristön kannalta keskeisiin laitteisiin ja niiden ominaisuuksiin. Luvussa esitellään ehdolla ollut Groma-robotti, toteutukseen käytetty Turtlebot II eri sensorivariaatioineen ja lopuksi käyttämämme TP-Linkin valmistama WLAN-tukiasema.

2.1 ROS

ROS (Robot Operating System) on robottien ohjelmointia helpottamaan tarkoitettu avoimen lähdekoodin ohjelmistoalusta. Määritelmänsä [10] mukaan se tarjoaa tyypillisiä käyttöjärjestelmäpalveluita, kuten muun muassa toteutuksen usein käytetyille toiminnoille ja sanomien välityksen prosessilta toiselle. Lisäksi se tarjoaa ympäristön ohjelmien ajamiseksi usean tietokoneen järjestelmissä. ROSin ydin on BSD 3-clause lisenssin [11] alainen, kun taas yksittäiset yhteisön kautta julkaistavat ohjelmapaketit voivat olla muidenkin lisenssien alaisia [12].

ROS ei korvaa perinteistä käyttöjärjestelmää, vaan toimii sen rinnalla tarjoten kokoelman kirjastoja ja työkaluja robottisovellusten rakentamiseen. Kantavana ideana on koodin uudelleenkäytön tukeminen robotiikan tutkimuksessa ja kehityksessä [10]. Prosessien hajauttaminen eri koneisiin on ROSin parhaita puolia. Eri koneissa toimivat prosessit, joita ROSin yhteydessä kutsutaan *nodeiksi (2.1.3)*, ovat löyhästi kytkettyjä ja voivat yleensä olla käynnissä toisistaan riippumatta. Poikkeuksen tähän tekee master-prosessi, jonka olemassaolo vaaditaan kaikkien muiden prosessien toiminnalle ja kommunikoinnille – tästä enemmän luvussa 2.1.1.

ROS-nodet kommunikoivat keskenään muun muassa sanomien välityksellä. ROSin sanomanvälitysrajapinta onkin tulossa *de facto* -standardiksi robottiohjelmistojen sanomanvälityksessä [13, p. 3]. Tämä parantaa yhteensopivuutta muiden järjestelmien rajapintojen kanssa.

ROS tarjoaa valmiita kirjastoja eri ohjelmointikielille. Tällä hetkellä ylläpidon ja kehityksen fokus on C++ ja Python tuessa. Muita kevyemmin tuettuja ovat muun muassa Lisp, Java, Lua ja R [14]. Ohjeita ja liitännäisiä (plugin) löytyy monille kehitysympäristöille, esimerkiksi hyvin tunnetut Eclipse, QTCreator ja NetBeans ovat edustettuina [15].

ROSin kehittämisestä vastaavat sen käyttäjät. Yhdistävänä tekijänä toimii ROS.org -yhteisö, joka on kollektiivinen joukko tutkijoita, insinöörejä ja asian harrastajia. Yhteisön taustalla on Open Source Robotics Foundation [16], joka sponsoroi ja ylläpitää ROS.org -sivustoa. Yhteisön jäsenet antavat oman erikoisosaamisensa muiden käyttöön julkaisemalla ja ylläpitämällä tekemiään ohjelmapaketteja. Kuka tahansa voi liittyä yhteisöön ja alkaa julkaista omia tuotoksiaan tai osallistua olemassa olevien ylläpitoon ja kehitykseen.

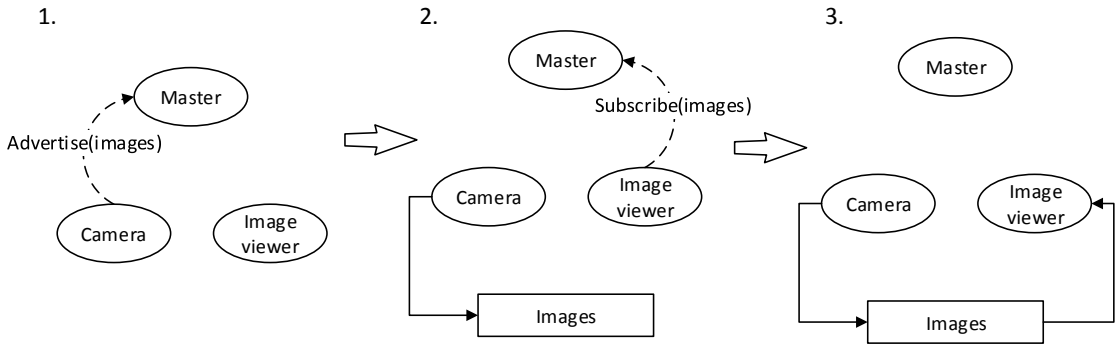
Luonnollisestikaan kehittäjät eivät julkaise yrityssalaisuudeksi jäävää kaupallisesti merkittävää aineistoa, joten parhaat toteutukset saattavat jäädä yleisen jakelun ulkopuolelle. Tutkimusrobotteihin, kuten Turtlebot II:een, löytyy valmiita käyttökelpoisia sovelluksia aina joystick-ohjauksesta kartoitukseen ja navigointiin asti.

Yleisesti ottaen ohjeistusta löytyy lähes kaikkeen, oikean ohjeen tai tiedon löytäminen vain voi joskus olla hieman työlästä. Näyttää siltä, että open source -maailmassa ylipäätään ohjeistus on usein hieman hajanaista, syynä lienee toiminnan talkooluonne. ROSia opetellessa kannattaakin tutustua heti alkuun aiheesta kirjoitettuun kirjallisuuteen ja kun käsitteet ovat selkeytyneet, hakea viimeisintä tietoa ROS.org-sivuilta.

2.1.1 Master

Master-node tarjoaa nimi- ja rekisteröitymispalvelut muille nodeille ja on edellytys näiden väliselle viestinnälle. Master pitää kirjaa siitä, mitkä nodet ovat rekisteröityneet mihinkin topiciin tai palveluun. Sen tehtävänä on auttaa nodeja muodostamaan P2P tyyppinen yhteys toisiinsa luovuttamalla näille tarvittavat yhteystiedot. [17, p. 38]

Yhteyden muodostamisen tyypilliset vaiheet (kuva 1):



Kuva 1. *Master yhdistää kaksi nodea nimeltään Camera ja Image_viewer topiciin ”images”, perustuu lähteeseen* [18]

1. Camera-node rekisteröityy (Advertise) masteriin ilmoittaen alkavansa julkaista kuvia ”images”-topicissa.
2. Camera on julkaisijana topicissa, mutta koska kukaan ei ole ko. topicin tilaajana, dataa ei vielä lähetetä. Jossakin vaiheessa Image_viewer rekisteröityy masteriin ”images”-topicin tilaajaksi (Subscribe).

3. Nyt topicilla on sekä julkaisija että tilaaja, jolloin master ilmoittaa molemmille nodeille toistensa olemassaolosta yhteystietoineen, jotta ne voivat jatkossa liikennöidä suoraan keskenään.

Master tarjoaa myös parametripalvelimen [17, p. 38]. Parametripalvelin ylläpitää ja jakaa ajonaikaisesti nodejen tarvitsemia parametritietoja. Sen käyttäjärajapinta on toteutettu XMLRPC-protokollalla [19], joka käyttää kutsujen koodaamiseen XML:ää ja tiedonsiirtoon HTTP:tä. [20]

Master käynnistyy komentoriviltä komennolla *roscore*, joka käynnistää samalla muut tarvittavat peruspalvelut. Master sammutetaan näppäinyhdistelmällä Ctrl-C. [13, p. 20]

2.1.2 Parametrit

ROS-ympäristössä parametreilla on tärkeä merkitys. Niillä voidaan muuttaa järjestelmän ja sen osasten toimintaa tarpeen mukaan. Tyypillisiä parametreja ovat esimerkiksi robotin maksiminopeus, maksimimittausetäisyys karttaan merkittävälle kohteelle tai robotin polun laskentaan käytettävä algoritmi ja sen algoritmikohtaiset parametrit – parametreja todellakin riittää moneen tarkoitukseen.

ROS-parametrit nimetään nodejen ja topicien nimiavaruuksien kanssa yhteensopivan hierarkkisen rakenteen mukaan. Tämä suojaa parametrien nimien yhteentörmäyksiltä, kun jokaista parametria kuvaa polku, joka sitoo sen omaan ympäristöönsä. Hierarkkisesta rakenteesta seuraa, että parametrit ovat saavutettavissa sekä erikseen että puuna. [20]

Esimerkkinä [20] kaksi rinnakkaista kameraa, joilla molemmilla on kaksi parametria – nimi ja valotusaika. Parametrit kameroille (vasen – oikea) voisivat olla:

```
/camera/left/name: leftcamera     /camera/right/name: rightcamera
/camera/left/exposure: 1          /camera/right/exposure: 1.1
```

Tässä esimerkiksi /camera/right/exposure identifioi valotusaika-parametrin ja 1.1 on sen arvo. Arvoja voidaan hakea siis paitsi suoraan em. parametrien nimillä, myös puuna. Voidaan esimerkiksi hakea molempien kameroiden tiedot, eli arvo polulle /camera, joka tässä tapauksessa olisi:

```
left: { name: leftcamera, exposure: 1 }

right: { name: rightcamera, exposure: 1.1 }
```

Vastaavasti voidaan hakea vain vasemman kameran tiedot, eli arvo polulle /camera/left, joka olisi vastaavasti:

```
name: leftcamera
```

```
exposure: 1.1
```

Parametreja voidaan asettaa käynnistystiedoston (launch file) välityksellä, ohjelmakoodissa tai komentoriviltä. Parametrien arvot voidaan myös kirjoittaa YAML-tiedostoon [21], joka voidaan sisällyttää (include) käynnistystiedostoon tai nimi antaa annettavan komentorivikomennon yhteydessä. Komentoriviltä parametreihin pääsee käsiksi komennolla *rosparam* (taulukko 1).

Taulukko 1. *rosparam-komennot* [22]

rosparam set	parametrin asetus
rosparam get	parametrin arvon haku
rosparam load	parametrien lataaminen tiedostosta[1]
rosparam dump	parametrien kirjoitus tiedostoon[1]
rosparam delete	parametrin poistaminen
rosparam list	parametrien nimien listaaminen

Esimerkkejä rosparam-komentojen syntaksista:

```
rosparam get /                                  // haetaan kaikki parametrit
rosparam set /camera/left/name leftcamera       // annetaan kameralle nimi
rosparam get /camera/left/name                  // haetaan kameran nimi
```

Esimerkki parametrin asetuksesta käynnistystiedostossa:

```
<param name="/camera/left/exposure" value="1.1" />
```

Esimerkki parametrin asetuksesta C++ koodissa:

```
ros : : param : : set ( "/camera/left/exposure " , 1.1) ;
```

Lisää parametrien asettamisesta ja lukemisesta komentoriviltä, käynnistystiedostossa ja C++ koodissa [13, pp. 105–115].

2.1.3 Node

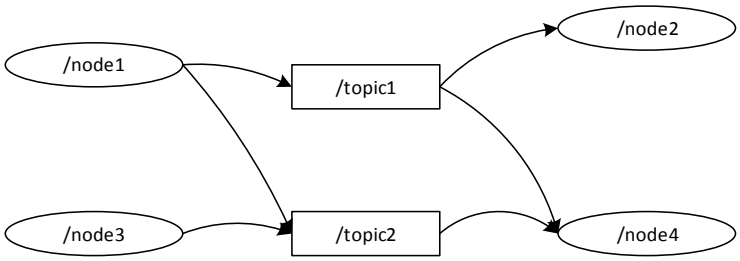
ROS-ohjelmisto rakentuu yleensä lukuisista *nodeista*, jotka voivat sijaita eri tietokoneissa ja kommunikoida verkon yli. Yksi node voi esimerkiksi hoitaa lasermittauksia, toinen taas voi käyttää sen tuloksia vaikkapa kartan tekemiseen, kolmas ehkä liikuttaa robottia kartan perusteella antamalla komentoja edelleen nodelle, joka ohjaa moottoreita ja niin edelleen.

[1] Tiedostomuoto YAML

Node on ohjelma, jota ajetaan yleensä *omana prosessina* ja joka suorittaa omia rajattuja tehtäviään. Poikkeuksen muodostaa noden sukulainen *nodeletti*, joka toimii muuten kuten yksittäinen node, mutta joita voi olla useita samassa prosessissa. Lähinnä C++:aa käytettäessä säästetään tällöin kapasiteettia sanomanvälityksessä, kun vältytään prosessien väliseltä liikenteeltä. Enemmän nodeleteista löytyy lähteestä [23].

Monimutkaisen järjestelmän hajottamisella löyhästi kytketyiksi erillisiksi nodeiksi saavutetaan muutamia etuja. Vikasietoisuus paranee, kun mahdollinen kaatuminen rajoittuu parhaassa tapauksessa vain yhteen nodeista. Koodin kompleksisuus vähenee ja ylläpito/kehitys helpottuu toteutuksen ollessa jaettuna tehtävän mukaisiin selkeisiin osiin. Lisäksi nodeja voidaan toteuttaa samaan kokonaisuuteen suhteellisen vapaasti esimerkiksi eri ohjelmointikielillä. [24]

Nodet muodostavat P2P-verkon ROS-prosesseja, jotka käsittelevät tietoa yhdessä [25]. Tätä kutsutaan laskentagraafiksi (kuva 2).



Kuva 2. *Yksinkertainen laskentagraafi.*

Tässä node1 on rekisteröitynyt julkaisijaksi kahteen topiciin ja node3 yhteen. Node2 on rekisteröitynyt tilaajaksi yhteen ja node4 kahteen topiciin.

2.1.4 Topic

Topicit ovat nimettyjä monta-moneen tyyppisiä sanomaväyliä, joiden avulla nodet välittävät sanomia toisilleen. Ne ovat vahvasti niissä julkaistaviin sanomiin tyypitettyjä, eivätkä voi ottaa vastaan muita sanomatyyppejä.

Nodet eivät tiedä, minkä noden kanssa ne kommunikoivat, vaan topicit erottavat ne toisistaan. Lähettävä osapuoli rekisteröityy masterille julkaisijaksi johonkin topiciin ja vastaanottava vastaavasti saman topicin tilaajaksi, kuten on jo esitetty luvussa 2.1.1. Yhteys on aina yksisuuntainen, eli vastauksia topicin julkaisija ei lähettämiinsä sanomiin saa. Kuten graafista (kuva 2) nähdään, samalla topicilla voi olla samanaikaisesti useampi julkaisija ja kuuntelija.

Tiedonsiirto tapahtuu käyttäen TCPROS- [26] ja UDPROS-protokollia [27], jotka puolestaan käyttävät standardeja TCP/IP- ja UDP-protokollia. ROSin omat headerit sisältävät ROS-spesifiä reititystietoa, joka voi sisältää muun muassa lähettävän noden tunnuksen,

siirtoon liittyvän topicin nimen, sekä käytettävän sanomatietotyypin. Oletusprotokollana on TCPROS; UDPROS on tätä kirjoitettaessa kehityksen alainen ja käytettävissä vain roscpp:ssä. [28]

Topiceihin pääsee käsiksi myös komentoriviltä komennolla rostopic. Seuraavassa muutama hyödyllinen rostopic-komento (taulukko 2):

Taulukko 2. *rostopic-komentoja.*

rostopic echo	tulostaa topicin sanomat
rostopic list	tulostaa aktiivisten topicien tiedot
rostopic pub	julkaisee dataa topiciin
rostopic type	tulostaa topicin sanomatyypin (sanoman määrittävä tietotyyppi)

Esimerkkejä rostopic-komennon syntaksista [29]:

```
rostopic echo /topicin_nimi
rostopic type /topicin_nimi
rostopic pub /topicin_nimi std_msgs/String moro, osta poro
```

(tässä std_msgs on paketti, josta sanoman tietotyyppi löytyy ja String on sanoma-tietotyyppi)

2.1.5 Sanoma

Sanoma on käytännössä rakenteinen tietotyyppi, jolla välitetään tietoa nodesta toiseen. Sanoma identifioidaan sen sisältävän paketin nimen ja sen kuvaustiedoston (tiedostotunniste .msg) nimen yhdistelmänä [30]. Otetaan esimerkkinä yksinkertainen sanoma, joka määrittelee robotin paikan ja orientaation kartalla:

```
polku sanoman määr. tiedostoon          sanoma
   turtlesim/msg/Pose2D.msg      ->     turtlesim/ Pose2D
```

Sanoman rakenteen saa helposti näkyviin komentoriviltä komennolla *rosmsg show*:

```
rosmsg show turtlesim/Pose2D

float64 x
float64 y
float64 theta
```

Nähdään sanoman koostuvan kolmesta kentästä, jotka määrittävät x- ja y-koordinaatit, sekä robotin orientaation eli ”rintamasuunnan” kyseisessä pisteessä. Kaikkien kenttien

arvot on esitetty 64-bittisellä liukuluvulla. Saman sanoman määrittelytiedosto Pose2D.msg:

```
# This expresses a position and orientation on a 2D manifold.

float64 x
float64 y
float64 theta
```

Sanomista voi joissakin jakeluissa tulla uusia versioita. Nodet voivat kommunikoida keskenään vain samaa versiota olevilla sanomilla, joten ne on voitava erottaa toisistaan. Sanomaversiot erotellaan MD5-tiivisteen avulla, joka kulkee ROS/Connect -headerissa. Tiiviste lasketaan sanoman määrittelytekstistä, jonka pelkät määrittelykentät huomioidaan laskennassa. [28], [31], [32]

2.1.6 Palvelu

Yksisuuntainen monelta-monelle tyyppinen julkaisija-tilaaja yhteys ei sovi esimerkiksi vastausta vaativia funktioiden etäkutsuja (RPC) varten. Tällaiset vuorovaikutukset hoidetaan ROSissa palveluilla. Palvelua käytettäessä asiakas lähettää palvelupyynnön, johon palvelun tarjoaja vastaa.

Palvelut määritetään sanomaparilla, toinen palvelupyyntöä ja toinen vastausta varten. Palvelulla on nimi – ja sen tarjoajaksi ja käyttäjäksi rekisteröidytään masterille – samaan tapaan, kuin topicinkin tapauksessa. [33]

Kuten sanoma, myös palvelu identifioidaan sen sisältävän paketin nimen ja palvelumäärittelytiedoston (tiedostotunniste .srv) nimen yhdistelmänä [34]. Esitetään tästä esimerkkinä itse määritelty esimerkkipalvelu:

```
polku palvelumäärittelytiedostoon                palvelu
omat_palvelut/srv/EsimPalvelu.srv   ->   omat_palvelut/EsimPalvelu
```

Palvelumäärittelytiedosto on samankaltainen, kuin sanomamäärittelytiedosto. Se vain sisältää kaksi katkoviivalla erotettua osaa, palvelupyyntösanoman ja siihen vastauksena saatavan sanoman sanomamäärittelyt.[17, p. 57]

Palvelumäärittelyn saa näkyviin komennolla *rossrv show*:

```
rossrv show omat_palvelut/EsimPalvelu

int32 A
```

```
int32 B
int32 C
---
int32 sum
```

Tässä on palvelupyyntösanomassa kenttä jokaiselle yhteenlaskettavalle ja vastaus-sanomassa kenttä summalle. Palvelun määrittelytiedoston sisältö olisi identtinen edellä olevan kanssa, tekijän mahdollisia kommentteja lukuun ottamatta.

Tietyn palvelun tarjoajia voi olla vain yksi, mutta käyttäjiä monta. Palvelun kaksisuuntaisen luonteen vuoksi se käyttää TCP/IP-pohjaista yhteyttä. [35]

2.1.7 Paketit

Paketti on ohjelmakokonaisuus, joka muodostaa loogisesti yhtenäisen moduulin. Tyypillisesti paketti käsittää muun muassa noden/nodeja, mahdolliset ROS-riippumattomat kirjastot, parametritiedostoja ja käynnistystiedostoja. Samaan kokonaisuuteen kuuluvat paketit voidaan koota yhteen, jolloin niistä muodostuu *stack*. Esimerkiksi navigation stack koostuu 17 paketista [36], joihin sisältyy muun muassa robotin paikantamiseen käytettävä AMCL (Adaptive Monte Carlo Localization), polun laskentaan käytettävä global_planner- ja robotin koko liikkumista kontrolloiva move_base -paketti.

ROSissa on roboteille mallikohtaisia paketteja, joissa on tietylle robottimallille räätälöityjä ominaisuuksia. Turtlebotillekin on asennettavissa omat pakettinsa, joista esimerkiksi turtlebot_navigation -paketti sisältää kartoitukseen ja navigointiin valmiit demo-käynnistystiedostot Turtlebotille sopivin parametriasetuksin. Nämä käynnistystiedostot puolestaan käyttävät muun muassa yleisiä edellisessä kappaleessa mainittuja paketteja.

2.1.8 Bag

ROS-Bag on debuggaustyökalu, jonka avulla nodejen lähettämät sanomat voidaan ottaa talteen halutuista topiceista ja kirjoittaa ne tiedostoon (tiedostotunniste .bag). Työkalun avulla voidaan tallentaa sensoridataa, tai tallennettu sanomaliikenne voidaan toistaa samalla tavalla ajoitettuna uudelleen, eli samoihin topiceihin julkaistaan samat sanomat samalla syklillä. [17, pp. 37, 96–102].

Bag-tiedostoa toistaessa voidaan valita, minkä topicien sanomat siitä julkaistaan [17, p. 97]. Tästä on erityisesti hyötyä silloin, kun halutaan testiympäristössä korvata jonkin bagistä toistettavan noden sanomat sen testiversion lähettämillä. Testaamisen ja vikakorjausten kannalta on erittäin hyödyllistä voida testata samoja skenaarioita moneen kertaan, mahdollisesti testattavan noden koodia lokittaen ja välillä muuttaen.

2.2 ROS 2

ROSista on julkaisuvaiheessa myös uusi versio. ROS 2:ssa CI-testaus (Continuous Integration) ja ylläpito laajenevat kattamaan Ubuntun lisäksi myös OS X:n ja Windowsin, mikä varmasti on tervetullutta käyttäjille, jotka ovat tähän asti joutuneet ajamaan ROSia virtuaalikoneilla. [37]

ROS 2:ssa käytetään tiedonsiirtoon DDS:ää (Data Distribution Service), josta on valittavissa muutama hieman erilainen versio [38]. DDS ei vaadi toimiakseen master-nodea, joka jääkin uuden ROS-version myötä historiaan. Uudesta tiedonvälitystavasta saadaan etua muun muassa monirobottijärjestelmiä silmällä pitäen. ROS 1:ssä ne on jouduttu eri keinoin virittelemään master-noden ympärille esimerkiksi multi-master -järjestelmää käyttäen [39], ja kun näitä virityksiä ei enää tarvita, ROS-monirobottijärjestelmistä tulee paljon yksinkertaisempia. [40]

DDS:n myötä voidaan varautua paremmin myös huonolaatuisen verkkoyhteyden aiheuttamiin ongelmiin. Se on oletuksena toteutettu UDP:n päälle, mutta implementoi myös TCP:n kaltaista yhteyden luotettavuus-toiminnallisuutta. Jotkin jakelijat tosin toimittavat nykyisin myös TCP:tä tukevia DDS-versioita, mutta toimiakseen se ei vaadi alleen kuin UDP:n toiminnallisuuden. DDS tuo ROSin tiedonvälitykseen QoS:n (Quality of Service), eli siirron ominaisuuksia voidaan säätää UDP:n ja TCP:n ominaisuuksien ”välimaastossa” tarpeen mukaan, pientä latenssia tai luotettavuutta painottaen. [40]

Myös pienten sulautettujen alustojen, kuten mikrokontrollereiden, tuki paranee DDS:n myötä. Niissä voidaan käyttää minimalistista protokollapinoa, kun jotkin jakelijat tarjoavat DDS:n standardi-rajapintaa suoremman pääsyn sen alemman tason wire-protokollaan RTPS:ään (Real-Time Publish-Subscribe) [41], jolloin koko DDS:n toiminnallisuutta ei niissä välttämättä tarvitse toteuttaa. [40],[42, p. 8]

ROS 2:sta tätä kirjoitettaessa julkaistu Alpha 7 release vaatii uudemman Ubuntu Xenial Xerus (16.04) version [43]. Koska uusi ROS-versio on vielä keskeneräinen ja voimakkaassa kehitysvaiheessa, tässä työssä pitäydytään edelleen ROS 1:ssä. Mikäli uudempaan ROS-versioon siirtyminen esimerkiksi monirobottiympäristön vuoksi katsotaan myöhemmin tarpeelliseksi, siihen on saatavissa migraatio-ohje [44].

2.3 Hadoop

Tietoa tuotetaan maailmassa lisää koko ajan kiihtyvällä vauhdilla, niinpä sen säilytys ja käsittely vaativat yhä enemmän resursseja. Puhutaan Big datasta, joka tarkoittaa valtavia määriä tietoa, joka on hankalasti käsiteltävissä perinteisin tietotekniikan menetelmin. Tällaista tietoa on esimerkiksi monimuotoinen tai ei-rakenteinen data, jonka käsittelyyn perinteiset tietokannat eivät kovin hyvin sovellu.

Big datalle on ominaista valtavan tietomäärän kertyminen suurella nopeudella ja se, että tästä tietomassasta pitäisi saada analysoitua tietoa nopeasti, ennen kuin se menettää arvonsa. Tätä ongelmaa on lähdetty ratkaisemaan muun muassa hajauttamalla ja rinnakkaistamalla tiedon käsittelyyn liittyviä toimintoja. [45]

Hadoop on avoimeen lähdekoodiin (Apache 2.0 lisenssi [46]) perustuva ohjelmistokehys, joka koostuu useista projekteista. Sen tunnetuimmat osat ovat HDFS ja MapReduce. HDFS tarjoaa tietomassojen luotettavan hajautetun tallennuksen ja MapReduce rinnakkaislaskentaan perustuvan tehokkaan tiedonkäsittelyn [47, pp. 12–14].

Tiedon tallennuksen ja laskennan hajauttaminen useille koneille antaa tehon lisäksi mahdollisuuden käyttää edullisempaa peruskalustoa järjestelmän osina. Hadoopissa luotettavuus sekä laskennassa että tallennuksessa saadaan aikaan redundanttisuudella. Laskennassa vikaantuneen koneen tehtävät voidaan antaa toiselle [47, pp. 200–204] ja tallennettava data sijoitetaan tyypillisesti kolmena kopiona eri koneille [47, p. 48].

Kun ROSissa kutsutaan nodeiksi prosesseja, Hadoopin yhteydessä nimitystä käytetään palvelimista. Laskentaan ja tallennukseen käytettävät palvelimet muodostavat *Hadoop-klusterin.* Järjestelmä on skaalautuva ja klusteriin on helppo lisätä tai siitä poistaa palvelimia tarpeen mukaan.

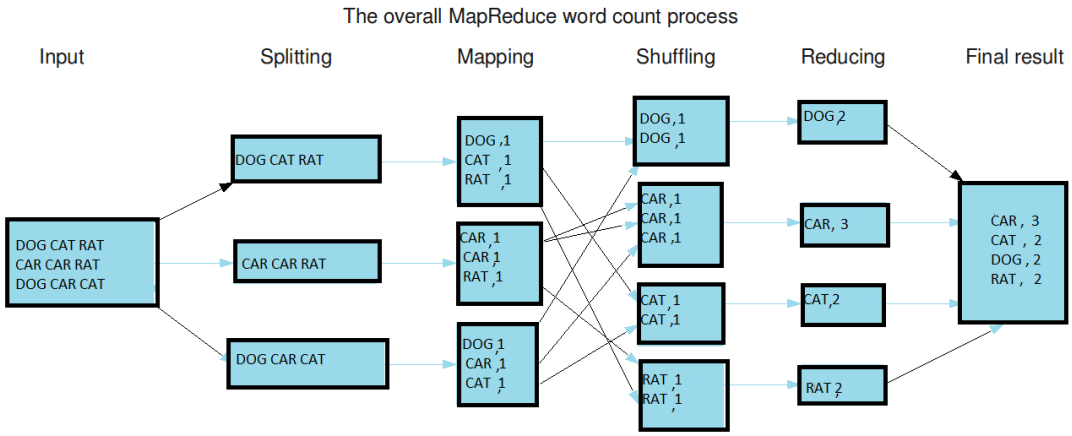
2.3.1 MapReduce

MapReduce on ohjelmistokehys, joka tarjoaa suhteellisen helpon tavan ohjelmoida sovelluksia suurten tietomäärien käsittelyyn. Laskennassa käytetään hyväksi joukkoa palvelimia, jotka kuuluvat Hadoop-klusteriin. Klusterissa tietoa käsitellään rinnakkaisesti ja näin saavutetaan suuria käsittelynopeuksia. [48]

Periaatteena on, että käsiteltävä tieto on sijoitettuna saman klusterin hajautettuun tiedostojärjestelmään HDFS:ään (2.3.2), ja että tämän datan käsittelyssä hyödynnetään ensisijaisesti samaa palvelinta, johon se jo on tallennettuna. Näin toimimalla vältytään massiiviselta tiedonsiirrolta palvelimien välillä. [48]

Yleisesti ottaen MapReduce jakaa tiedon sopiviksi toisistaan riippumattomiksi osiksi, jotka syötetään rinnakkaisille map-taskeille prosessoitaviksi. Näiden tuottama data lajitellaan ja syötetään edelleen reduce-taskeille, jotka kokoavat tiedon sopivaan ulostulomuotoon – tästä nimi MapReduce. Hadoop hallinnoi taskien suoritusta ja allokoi epäonnistuneet uudelleen. [48]

Esimerkkinä MapReduce-työn etenemisestä lasketaan sanojen esiintymismäärä syötteenä annetussa tekstissä (kuva 3).



Kuva 3. *Hadoop MapReduce, toimintaperiaate* [49].

Aluksi järjestelmään syötetty data (input) jaetaan map-taskeille (splitting), joita voi samassa palvelimessa olla joskus useitakin. Map-taskit muodostavat jokaisesta löytämästään sanasta avain-arvoparit (mapping), joissa avaimena toimii tässä sana ja arvona 1.

Seuraavaksi saman avaimen sisältävät parit lajitellaan yhteen (shuffling) ja edelleen reduce-taskit laskevat lajiteltujen avain-arvoparien arvojen summan, tuottaen uudet avain-arvoparit (reducing), tällä kertaa avaimena sana ja arvona em. summa eli sanojen lukumäärä.

Lopuksi reduce-taskien ulostulot kirjataan tekstitiedostoon, jossa on lopputuloksena kaikkien sanojen esiintymismäärät (final result).

Tässä työssä ei hyödynnetä MapReducea, mutta Hadoop-asennuksen myötä sekin on OpenCRP-ympäristössä käytettävissä, mikäli se jatkossa tarpeelliseksi nähdään. Tämän vuoksi sen käsittely jää tässä varsin pintapuoliseksi.

2.3.2 HDFS

HDFS on nimensä mukaisesti hajautettu tiedostojärjestelmä, joka toimii Hadoop-klusterissa. Sen tarkoitus on hajauttaa suurten tietomäärien tallennusta sekä varmuuden että tehokkuuden nimissä. Varmuus siis tulee redundantista tallennuksesta ja tehokkuus MapReduce-aineiston sijoittamisesta laskentapaikkaan, kuten jo aiemmin on todettu.

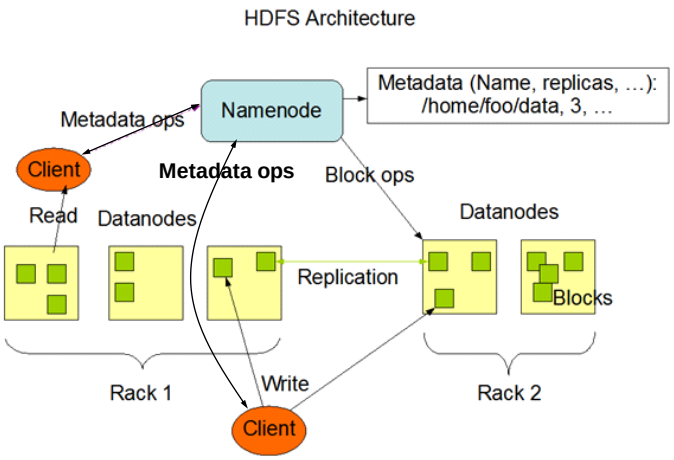
Tallennusblokin oletuskoko on 128 MB, tätä suuremmat tiedostot jaetaan useampaan blokkiin. Jos tiedosto on niin suuri, että se on paloiteltava, sen blokit pyritään järjestämään niin, että ne sijaitsevat fyysisesti eri koneissa. Kun käytetään suurta määrää peruskoneita tiedon tallentamiseen, myös vikojen todennäköisyys koko järjestelmässä kasvaa suureksi. Tämän vuoksi jokainen blokki replikoidaan monelle erilliselle koneelle, oletus-

arvoisesti kopioita jokaisesta blokista on kolme. Kopioiden määrää voidaan myös kasvattaa suositun tiedoston blokeille ja näin tasata sen samoihin blokkeihin kohdistuvaa yhtäaikaista lukukuormaa useammalle koneelle. Sekä blokin koko että kopioiden määrä on haluttaessa asetettavissa tiedostokohtaisesti ja niiden muuttaminen on mahdollista myös tallennuksen jälkeen. Mikäli blokki jostain syystä menetetään, se replikoidaan muista kopioista edelleen uudelle koneelle, jotta kopioiden määrä (= replication factor) pysyy samana.

HDFS:ssä on kahdentyyppisiä palvelimia: niminode (namenode) ja datanode (datanode), jotka toimivat master-slave periaatteella. Ne ovat käytännössä ohjelmia, jotka on suunniteltu toimimaan tavallisissa työasemissa. Niminodeja on yksi ja mahdolliset varalla olevat, tiedon tallentajina toimivia datanodeja taas on skaalautuva määrä. Niminode hallinnoi koko tiedostojärjestelmää ja sen tiedossa on, missä datanodessa minkäkin tiedoston blokit sijaitsevat. HDFS on kirjoitettu Javalla ja missä tahansa palvelimessa, joka tukee Javaa, voidaan ajaa nimi- tai datanodea. Käytännössä yhdessä koneessa ajetaan yhtä nimi- tai datanodea. [50]

Koska niminode on avainasemassa koko järjestelmän kannalta, sen toiminta on varmennettava. Tähän voidaan käyttää useampiakin menetelmiä, kuten tiedostojärjestelmän metadatan redundanttia tallentamista ja sekundäärisiä tai stand-by niminodeja. [47, pp. 47–51, 338–339].

Kuvassa 4 esitetään HDFS:n arkkitehtuurikaavio. Tässä ”Rack” tarkoittaa laitekaappia/-räkkiä, jossa on useampia koneita sijoitettuna samaan kehikkoon. *Client* on tietokone, jolta HDFS-palveluja käytetään. Siinä on Hadoop asennettuna, mutta se ei ole master, eikä slave Hadoop-klusterissa. *Metadata* pitää sisällään tietoa tallennetusta tiedosta, kuten tiedostonimen, kopioiden lukumäärän ja niiden sijainnit. Metadata ops käsittää metadataan perustuvia pyyntöjä, kuten tiedostojen tallennuspaikkatietojen pyynnöt ja niihin liittyvät vastaukset. *Block ops* pitää sisällään esimerkiksi tiedoston poistamisen tai replikoinnin käynnistämisen vian seurauksena.



Kuva 4. *HDFS arkkitehtuuri, perustuu lähteeseen* [50].

Tiedoston tallentaminen tapahtuu seuraavasti [50]:

Kun sovellus haluaa tallentaa dataa uuteen tiedostoon, järjestelmä ohjaa kirjoittavalle sovellukselle näkymättömästi sen sisällön ensin paikalliseen väliaikaistiedostoon. Kun tämä tiedosto kasvaa yli yhden blokin koon, client ottaa yhteyden niminodeen (Metadata ops), joka lisää tiedostonimen tiedostojärjestelmään ja allokoi sille datablokin. Niminode vastaa clientille identifioimalla tälle listana datanodet tallennusta ja replikointia varten, sekä sille allokoidun datablokin (Metadata ops). Client siirtää saamiensa tietojen perusteella paikallisen väliaikaistiedoston sisältämän datan sille allokoituun blokkiin datanodessa (Write). Blokkien tallennuksia niminoden aina blokkikohtaisesti jakamiin paikkoihin jatketaan, kunnes tiedosto suljetaan ja loppu väliaikaistiedoston data siirretään datanodeen. Client kertoo lopuksi niminodelle, että tiedosto on kirjoitettu loppuun (Metadata ops), jolloin tämä päivittää luodun tiedoston tilan valmiiksi tiedostojärjestelmään.

Blokin lähetyksen yhteydessä client toimittaa myös saamansa replikointilistan sille datanodelle, johon se blokin tallettaa. Tämä ensimmäinen datanode replikoi blokin edelleen listan seuraavalle datanodelle (Replication) välittäen sille myös replikointilistan ja tämä ketju jatkuu, kunnes tarvittava määrä kopioita on saatu aikaan.

Datanodet vastaanottavat datan pienemmissä erissä ja aina erän saavuttua ne alkavat jo replikoida ko. erää eteenpäin. Tieto siis ”valuu” jo kirjoitusvaiheessa eteenpäin datanodeketjussa. Normaalisti kopioiden määrän ollessa kolme, HDFS tallentaa kaksi ensimmäistä blokin kopiota saman räkin eri nodeihin räkkien välisen liikenteen minimoimiseksi ja kolmannen kopion toiseen räkkiin, jotta esimerkiksi räkin virransyötön katkeaminen ei veisi kaikkia kopioita.

Tiedoston lukeminen [47, pp. 69–72]:

Client pyytää niminodelta tiedoston muutaman ensimmäisen blokin tiedot. Niminode palauttaa blokkien id:n lisäksi niiden datanodejen osoitteet, joissa kukin blokki sijaitsee (Metadata ops). Lisäksi datanodejen osoitteet on lajiteltu clientia lähimmästä nodesta kauimpaan (oma kone, sama räkki, sama datacenter jne). Blokkien sijainnit saatuaan client lukee blokit datanodeilta (Read) ottaen ensin yhteyden lähimpään ensimmäisen blokin sisältävään datanodeen lukemista varten, sitten seuraavan blokin lähimpään nodeen jne. Mikäli luettaessa tapahtuu virheitä, client yrittää uudestaan aina seuraavaksi lähintä datanodea. Itse tietoa lukeva sovellus ei näe näitä vaiheita, vaan sille luku näkyy vain tietovirtana, kuten mitä tahansa tiedostoa lukiessa. Tarpeen vaatiessa client hakee nimipalvelimelta aina lisää metadataa ja jatkaa kunnes koko tiedosto on luettu.

Nimipalvelin valvoo datanodejen toimintaa muun muassa siten, että jokainen datanode lähettää määrävälein ns. ”heartbeat”-sanoman nimipalvelimelle. Jos jokin datanode ei lähetä sanomaa ajallaan, nimipalvelin päättelee sen olevan poissa toiminnasta ja käynnistää replikoinnin sillä olevien datablokkien replikointiarvon pitämiseksi oikeana. [50]

HDFS:n kommunikaatioprotokollat toimivat TCP/IP-kerroksen päällä. Client ja datanodet käyttävät omia protokolliaan (ClientProtocol ja DataNode Protocol), jotka on abstrahoitu käyttäen RPC-kutsuja. Politiikkana on se, että kommunikoinnin aloittaa aina client tai datanode, ei koskaan niminode. [50]

2.4 Qt

Qt on alustariippumaton kehitysympäristö pöytäkoneille, mobiililaitteille ja sulautetuille järjestelmille. Qt:n ylläpidosta ja lisensoinnista vastaa Qt Company Ltd [51]. Qt Company myy siihen kaupallisia lisenssejä, mutta se on saatavissa myös open sourcena osittain GPL- ja osittain LGPL-lisenssiversioiden alla. Qt:n lisensoinnista enemmän lähteessä [52].

Qt itsessään on koodattu C++:lla ja laajentaa sitä omilla lisäominaisuuksillaan, jotka parseroidaan ennen käännöstä ja generoidaan standardina C++:na. Tästä seuraa, että varsinainen käännös voidaan tehdä käyttäen mitä tahansa standardia C++ kääntäjää, kuten vaikkapa GCC (Linux) tai MinGW (Windows). [53]

Qt:n mukana jaetaan Qt Creator, joka on graafinen ympäristö ohjelmointiin ja graafisten käyttöliittymien suunnitteluun ja toteutukseen. Sitä voidaan käyttää Linuxin, Windowsin ja OS X:n päällä. Graafisten käyttöliittymien perusrakennuspalikat löytyvät valmiina sen valikoimista, painonapeista tekstikenttiin ja valikoihin. Niiden asettelu ja muokkaaminen on vaivatonta Creatorin sisältämässä Qt Designer-työkalussa. Designerin käyttö jättää enemmän aikaa varsinaiseen toiminnallisuuteen keskittymiseen, kun ulkoinen ilme saadaan nopeasti halutuksi ja muutosten teko siihen käy helposti.

Myös ROS tukee Qt:tä, ROSissa voidaan luoda Qt-pakettitemplate, joka sisältää valmiiksi ”hello world”-tyyppisen ROS-noden ja siihen liittyvän käyttöliittymän. Tästä enemmän luvussa 3.6, jossa esitellään toteutettu testikäyttöliittymä.

2.5 Versionhallinta

Robotin testikäyttöliittymän kehitystä varten tarvittiin versionhallintatyökalu. Versionhallinnan tarkoitus on pitää ohjelmiston osat tallessa siten, että sen kehitysvaiheet ovat jäljitettävissä myöhemmin. Tyypillisesti aina, kun saadaan jokin ominaisuus valmiiksi ja ohjelmasta on kääntyvä ja toimiva versio, kaikki edellisestä tallennuskerrasta muuttuneet lähdetiedostot tallennetaan versionhallintaan. Aina, kun tiedosto tallennetaan, siihen tehdyt muutokset merkitään historialokiin ja siitä tulee uusi versio sen evoluutioketjuun.

Yleisesti tähän tarkoitukseen käytetty GitHub-pilvipalvelu [54] on maksuton vain, jos sinne tallennettu repository (projektin tiedostojen tallennuspaikka) on julkinen. Tässä vai-

heessa yksityinen repository tuntui paremmalta vaihtoehdolta, kun kyse on testisovelluksesta omaan käyttöön, eikä kaikkia tulevaisuuden versioihin liittyviä komponentteja tai niiden lisensointia vielä tunneta.

GitLab [55] taas on versionhallintaohjelmisto, jonka voi asentaa omalle koneelle. Siitä on saatavilla sekä kaupallinen että open source -versio. Se on kehitetty 64-bittisille Unix käyttöjärjestelmille, eikä se toimi suoraan esimerkiksi Windowsin päällä [56]. GitLabin verkkosivuilla on lyhyt asennusopas [57], jonka avulla palvelin on suhteellisen helposti asennettavissa. GitLabin peruskäyttö Qt-työasemalta ei käytännössä eroa GitHubin käytöstä.

2.6 Groma

iRobotin Groma iRobot Create on lähinnä kartoitustarkoituksiin tarkoitettu ohjelmoitava mobiilirobotti, joka julkistettiin jo vuonna 2007 [58]. Siitä on tätä kirjoitettaessa tullut uusi versio iRobot Create 2 [59], johon emme ole käytännössä päässeet tutustumaan.

Groman (Create 1) paikannus perustuu lähinnä pyörien pyörimisen mittaamiseen, vaikka se osaakin korjata sijaintiaan tunnetuissa kohteissa. Esteiden havaitseminen tapahtuu etupuskurin taakse sijoitetuilla kytkimillä, joita on yksi molemmilla sivuilla ja keskellä. Alun perin Gromaa (kuva 5) oli tarkoitus käyttää toimintaympäristön kartoittamisessa, mutta heti alkuvaiheessa sen toiminnassa havaittiin tarkoitukseemme liiallista epätarkkuutta.

Kuva 5. *Groma iRobot Create.*

Käyttämällämme suhteellisen kovalla ja liukkaalla lattiapinnalla sen pyörät pyörivät tyhjää ja liukuivat helposti. Kun laite otti vinottain kontaktin kiinteään esteeseen, menetti se samalla hieman pyörähtäessään suuntavaistonsa siinä määrin, että muodostetuista kartoista ei tullut käyttökelpoisia.

Vaikka laitteen ohjauskeskuksena toimivassa Command Center -ohjelmassa olikin mahdollista valita alustan materiaali, tämä ei juuri tilannetta parantanut. Niinpä päätimme

luopua Groman käytöstä ja korvata sen toisella Turtlebotilla. Groman tekniset tiedot on esitetty taulukossa 3.

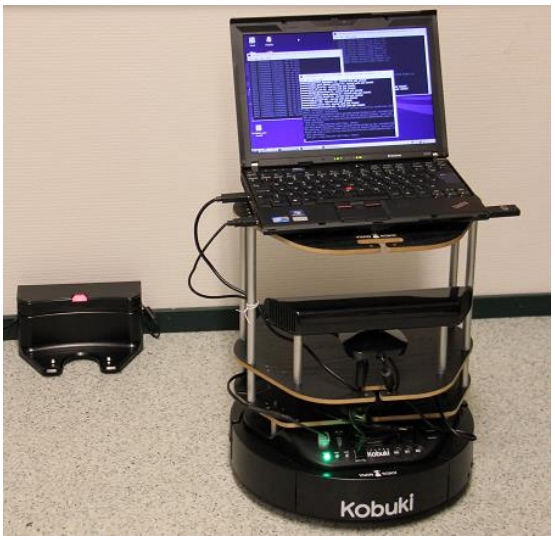
Taulukko 3. *Groman tekniset tiedot*

Max nopeus	50 cm/s
Toiminta-aika	≤ 3,5 h (tehoakku)
Latausaika	3 h (tehoakku)
Telakointi	ympärihavaitsevan IR-sensorin avulla
PC kytkentä	Bluetooth-moduuli (lisävaruste)/mini-din -sarjaportti
Puskuritunnistin	vasen, oikea
Porrastunnistimet	vasen, vasen-keski, oikea-keski, oikea
Pyörän pudotus-sensorit	vasen, oikea
Seinän läheisyys-sensori	oikea etukulma
Jänniteulostulot	5V/100 mA, 3 * 14,4 V/1.5A (akkujännite)
Latauksen LED-ilmaisin	kaksivärinen LED [hidas vilkkuva oranssi – lataus (nopea = elvytyslataus), vihreä – täysi, oranssi – osittain purkautunut, punainen – lähes tyhjä, vilkkuva punainen - tyhjä]
Käyttökytkimet	3 kpl
Akku	12 * AA alkaliparisto/iRobot Roomba -akut/14,4 V 3300 mAh tehoakku
Telakoinnin IR-tunnistin	yksi ympärihavaitseva

[60], [61], akun tyyppikilpi

2.7 Turtlebot

Käyttöömme oli jo aiemmin hankittu korealaisen Yujin Robotin Kobuki Turtlebot II -mobiilirobottialusta. Se on Yujin Robotin ja amerikkalaisyhtiö Willow Garagen yhteistyönä kehitetty halpa ROSia tukeva robottialusta [62], joka julkistettiin ROS-kehittäjien konferenssissa (ROSCon) v. 2012 [63]. Tämä alusta on tarkoitettu tutkimus- ja opetuskäyttöön ja on varustettu monipuolisilla antureilla, liitännöillä ja lataavalla telakointiasemalla (kuva 6).



Kuva 6. *Latausasema ja Turtlebot II.*

Robotin päälle sijoitettua kannettavaa tietokonetta voi ladata robotin kautta 19V jännitteellä ja max. 2.1 A virralla. Tosin tämä latausliitäntä antaa virtaa vain silloin, kun robotti

on telakoituneena. Alustan 5V/1A ja 12V/1.5 ja 5 A liitännät sen sijaan ovat käytettävissä myös muulloin. Laitteen tärkeimmät tekniset tiedot on esitetty taulukossa 4.

Taulukko 4. *Turtlebot II:n tekniset ominaisuudet* [64, pp. 3–4].

Max nopeus	65 cm/s
Max kääntymisnopeus	3.14 rad/s
Max kuorma	5 kg (kova lattia), 4 kg (matto)
Portaan laskeutumiskyky	ei aja > 5 cm syvyisen reunan yli
Kynnyksen ylitys/ matolle nousu	≤ 12 mm
Toiminta-aika	3/7 h (pieni/suuri akku)
Latausaika	1.5/2.6 h (pieni/suuri akku)
Telakointi	osaa telakoitua 2 m * 5 m (lev.*pit.) alueella telakointiaseman edessä
PC kytkentä	USB tai RX/TX-pinnit rinnakkaisportissa
Moottorin ylikuormitussuoja	katkaisee virran moottoreille ylikuormitustilanteessa
Odometria	25718.16 mitt./pyörän kierros, 11.7 mitt. /mm
Gyroskooppi	tehdaskalibroitu, 1 akselinen (100 °/s)
Puskuritunnistin	vasen, keski, oikea
Porrastunnistimet	vasen, keski, oikea (ks. portaan laskeutumiskyky)
Pyörän pudotus-sensorit	vasen, oikea
Jänniteulostulot	5V/1A, 12V/1.5A, 12V/5A
Tilan LED-ilmaisin	kaksivärinen LED [vilkkuu – lataus, vihreä – korkea varaustaso, oranssi – matala varaustaso]
Painokytkimet	3 kpl ohjelmoitavia painikkeita
Akku	14.8 V Li-Ion 2200 mAh (pieni) 4400 mAh (suuri)
Sensorien mittaustaajuus	50 Hz
Laturi	Input: 100-240V AC, 50/60Hz, 1.5A max Output: 19V DC, 3.16A
Kannettavan latausliitäntä	19V/2.1A DC
Telakoinnin IR-tunnistimet	vasen, keski, oikea

Jouduttuamme luopumaan Gromasta, tilasimme toisen Turtlebot II:n. Tietokoneena molempien alustojen päällä meillä on Lenovo X201:t (kuva 6 yllä). Koneet on varustettu SSD-levyasemilla (Solid-State Drive), jottei tärinä aiheuttaisi ongelmia kiintolevyille.

2.7.1 Kinect for Xbox 360

Toinen käytössämme oleva Turtlebot II on varustettu Kinect for Xbox 360 -sensorilla. Se on Microsoftin Xboxia varten kehittämä ja v. 2010 julkistama [65] 3D-sensori. Sensorin tärkeimmät osat ovat IR-projektori, RGB kamera ja IR-kamera (kuva 7, em. vasemmalta lukien).

Kuva 7. *Kinect 3-D sensori.*

Hieman yllättäen Microsoftin sivuilta ei löydy Kinectin Xbox 360 -mallille tarkempia teknisiä tietoja. Sensorin ominaisuuksista on eri lähteissä hieman ristiriitaista tietoa, koska varsinkin havaintoetäisyys vaihtelee eri ohjelmistoilla. Seuraavassa robotin meille toimittaneen yrityksen sivustolla esitetyt tekniset tiedot (taulukko 5), poikkeuksena tarkennetut tiedot havaintoetäisyydelle lähteestä [66] ja mikrofonien määrälle lähteestä [67].

Taulukko 5. *Kinect for Xbox, tekniset ominaisuudet* [68].

Tehon kulutus	12 W (tarvitsee 12 V ulkoisen virtalähteen)
Havaintoetäisyys	0.8 … 4 m [66]
Näkökenttä	57° H/43° V (hor./vert.)
Sensorit	RGB & syvyys & mikrofonisuora (4 mikrofonia [67])
Syvyyskuvan koko	VGA (640 * 480) : ~9 – 30 Hz
Resoluutio	max 1280 * 1024
Alusta	Intel X86 & AMD
Käyttöjärjestelmät	Win 7, Linux Ubuntu (X86,32/64 bit)
Liitäntä	USB2.0
Ohjelmisto	software development kits(OPENNI SDK bundled)
Ohjelmointi	C++/C#, Visual Basic .NET(Windows)
Toimintaympäristö	sisätilat
Mitat	30.7 * 6.4 * 5 cm

Taulukossa on mainittu mittausetäisyytenä Xbox-käytössä pätevä mittausalue. Käyttämällämme ROSin OpenKinect/libfreenect -ohjelmistolla on päästy selvästi edellä mainittua parempiin tuloksiin [69] mittausalueen ollessa tällöin ~ 0.5… > 8 m.

2.7.2 Xtion Pro Live

Toinen roboteistamme varustettiin v. 2012 julkaistulla [70, p. 19] taiwanilaisen Asuksen Xtion Pro Live sensorilla (kuva 8). Alun perin tilasimme robottiin Orbbec Astran, mutta sen ajureiden kehitys oli kuitenkin toimittajan mukaan vielä kesken. Tämä olisi aiheuttanut meille ylimääräisen työmäärän riskin, joten valitsimme varmuuden vuoksi jo pitkään käytössä olleen Xtion Pro Liven.

Kuva 8. *Xtion Pro Live*

Sensori makaa vaakasuoran levyn päällä, johon se kiinnitetään vain kevyen oloisella sensorin runkoon valetulla muovihaarukalla. Haarukka viedään levyssä olevasta reiästä läpi ja kiristetään pultilla ja mutterilla reiässä olevien ulokkeiden ympärille. Koska haarukkaa

ei voi vaihtaa, sitä kiristettäessä on syytä olla varovainen ja muistaa laittaa mukana seurannut suojaholkki sen väliin. Sensorin tekniset tiedot on listattu taulukossa 6.

Taulukko 6. *Xtion Pro Live, tekniset tiedot* [71].

Tehon kulutus	< 2.5 W
Havaintoetäisyys	0.8 … 3.5 m
Näkökenttä	58° H/45° V/70° D (hor./vert./diag.)
Sensorit	RGB & syvyys & 2 mikrofonia
Syvyyskuvan koko	VGA (640*480): 30 fps QVGA (320 * 240): 60 fps
Resoluutio	SXGA (1280 * 1024)
Alusta	Intel X86 & AMD
Käyttöjärjestelmät	Win 32/64: XP, Vista, 7, 8 Linux Ubuntu 10.10: X86,32/64 bit Android (by request)
Liitäntä	USB2.0/ 3.0
Ohjelmisto	software development kits (OpenNI SDK bundled)
Ohjelmointi	C++/C# (Windows) C++(Linux) JAVA
Toimintaympäristö	sisätilat
Mitat	18 * 3.5 * 5 cm

Mainittu minimimittausetäisyys on tässäkin varsin suuri 0.8 m, mutta esimerkiksi lähteen [72] mukaan sillä on kyetty havaitsemaan 1 cm kokoinen objekti 0.6 m etäisyydeltä. Vertailun vuoksi sama ryhmä oli havainnut Kinectillä 2 cm kokoisen objektin 0.6 m etäisyydeltä, joten Xtion Pro Livessä näyttäisi olevan potentiaalia parempaankin (vrt. Kinect 2.7.1).

2.7.3 Orbbec Astra

Orbbec on juuriltaan kiinalainen, sittemmin Yhdysvaltoihin laajentunut v. 2013 perustettu 3D-kameroihin keskittynyt yritys. Orbbec Astra 3D-sensori (kuva 9) tuli markkinoille v. 2015 ja sillä on tätä kirjoitettaessa jo seuraajat Astra Pro ja Persee, joille haettiin yhteisörahoitusta samana vuonna. [73], [74]

Kuva 9. *Orbbec Astra* [75].

Astrassa on hieman lyhyempi minimihavaintoetäisyys, kuin kilpailijoilla. Toivoimme tästä olevan hyötyä robotin liikkuessa dynaamisesti ahtaassa ympäristössä, esimerkiksi

kun ihmisiä liikkuu aivan robotin lähietäisyydellä. Astran hankinta jätettiin kuitenkin tulevaisuuteen (2.7.2), mikäli tarpeelliseksi nähdään. Sen tekniset ominaisuudet esitellään taulukossa 7.

Taulukko 7. *Orbbec Astra, tekniset ominaisuudet* [75].

Mittausetäisyys	0.4 … 8 m
Syvyyskuvan koko	640 * 480 (VGA) 16bit @ 30FPS
RGB kuvan koko	1280*960 @ 10FPS
Näkökenttä	60° hor. x 49.5° vert. (73° diag.)
Liitäntä	USB 2.0
Mikrofonit	2
Käyttöjärjestelmät	Windows, Linux, Android
Käyttöjännitteen syöttö	USB 2.0
Ohjelmistot	Orbbec Astra SDK + OpenNI
Mitat	160 * 30 * 40 mm

Astran minimimittausetäisyys (0.4 m) olisi ollut varsin lupaava, mutta valitettavasti emme nyt päässeet käytännössä näkemään, mihin se pystyy. Siitä on myös lyhyemmille mittausetäisyyksille tarkoitettu malli Astra S, jonka mittausalue on 0.3 … 5.8 m [75].

Astra Pro ja Persee toivat v. 2016 RGB-kuvaan suuremman 30 FPS (Frames Per Second) kehysnopeuden hieman pienemmällä 720p (1280*720) resoluutiolla. Persee sisältää lisäksi muun muassa integroidun tietokoneen kaksiydinprosessorilla. [74], [76], [77]

Orbbecin tuotteista ei valitettavasti löytynyt luotettavaa testitietoa vielä tätä kirjoitettaessa. Sen uudet mallit ovat kuitenkin tutustumisen arvoisia projektin jatkoa ajatellen.

2.8 TP-Link Archer C2600

OpenCRP:ssä robotin tietokone yhdistetään verkkoon WLANilla. Käyttöömme hankittiin kaksi TP-Link Archer C2600-tukiasemaa (kuva 10), joista toisella jatkamme tarvittaessa tukiasemapeittoa suuremman toiminta-alueen aikaansaamiseksi.

Kuva 10. *Archer C2600* [78].

Näillä tukiasemilla korvattiin tietoliikennelaboratorion jo käyttöikänsä loppupuolelle ehtineet vanhemman 802.11 b/g/n -standardin tukiasemat. Archerin tärkeimmät tekniset ominaisuudet on esitetty taulukossa 8.

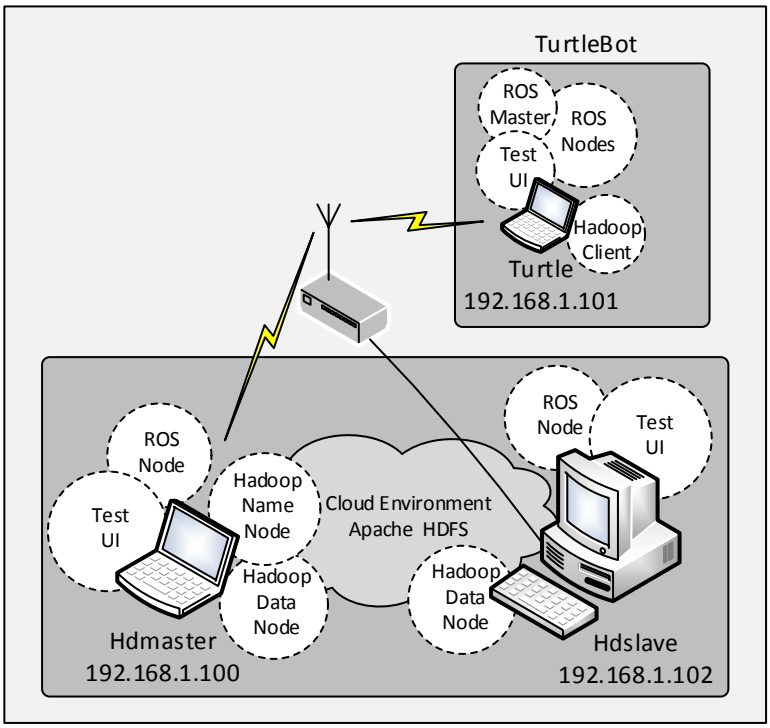
Taulukko 8. *TP-Link, tekniset ominaisuudet* [79].

Liitännät	4 * 10/100/1000Mbps LAN 1 * 10/100/1000Mbps WAN 2 * USB 3.0
Langattomat standardit	IEEE 802.11ac/n/a 5GHz IEEE 802.11b/g/n 2.4GHz
Taajuudet	2.4GHz ja 5GHz
Nopeudet	5GHz Band: Up to 1733Mbps 2.4GHz Band: Up to 800Mbps
Suojaus	64/128-bit WEP, WPA/WPA2, WPA-PSK/WPA-PSK2 encryptions
Antenni	4 * irrotettava antenni
Käyttöjännite	12V/4A
Mitat	263.8 * 197.8 * 37.3 mm

Archer C2600 tukee MU-MIMOa (Multi User Multiple-Input and Multiple-Output), joka on 802.11ac -standardin sisältämä siirtotekniikka, jota hyväksi käyttäen tukiasema kykenee jakamaan datavirtaa useammalle client-laitteelle samanaikaisesti. Vanhempi SU-MIMO-tekniikka (Single User MIMO) jakoi datavirran ajallisesti laitteiden kesken, kun taas tämä tukiasema kykenee palvelemaan maksimissaan 3:a laitetta samanaikaisesti. Tämä perustuu mahdollisuuteen suunnata laitekohtainen lähetyssignaalin osa ko. laitteen suuntaan antenneille syötettävien osasignaalien vaihe-eroja säätämällä. Suuntimista varten tukiasema lähettää toisistaan riippumattomia mittauskehyksiä jokaisesta antennistaan ja client-laitteet ilmoittavat, millä voimakkuudella ne kunkin kehyksen vastaanottivat. Tämän perusteella tukiasema laskee suuntimat kullekin laitteelle. Aiheesta enemmän lähteissä [80], [81] ja [82].

3. OPENCRP

Tässä luvussa esitellään OpenCRP-pilotointi- ja demonstraatioympäristön toteutus. Ympäristössä yhdistyy kolme tietokonetta, robotin (Turtlebot) tietokone Turtle ja kaksi etäkonetta Hdmaster ja Hdslave. Koneet on yhdistetty toisiinsa WLAN-reitittimen välityksellä. Hdslave kytkettiin reitittimeen kaapelilla, Hdmaster ja Turtle langattomasti. Jokaiselle koneelle on määritetty kiinteä IP-osoite. Ympäristön kokoonpano on esitetty kuvassa 11.



Kuva 11. *OpenCRP-pilotointi- ja demonstraatioympäristö*

Kaikissa koneissa on asennettuna Hadoop ja ROS, sekä testikäyttöliittymä (Test UI) robotin ohjausta varten. Testikäyttöliittymään sisältyy ROS-node (ROS Node), jonka kautta se kommunikoi ROSin kanssa.

Hdmaster ja Hdslave muodostavat kahden koneen HDFS-klusterin (Cloud Environment Apache HDFS). Robotin tietokone toimii Hadoop-clientina, ja siinä ajettavan client-applikaation (Hadoop Client) avulla voidaan tallentaa toimintaympäristöjen karttoja robotilta HDFS:ään, tai ladata niitä sieltä robotille.

Hdmaster on nimensä mukaisesti Hadoop-master -kone ja siinä ajetaan Hadoopin nimipalvelinta (Hadoop Name Node), joka hallinnoi HDFS-tiedostojärjestelmää. Hdmasterissa ja Hdslavessa ajetaan datanodeja (Hadoop Data Node), jotka toimivat tiedon tallentajina.

Vaikka robotteja on kaksi, niistä vain toista käytetään kerrallaan samassa verkossa. Robottien ajaminen on mahdollista myös ilman verkkoa, kun testikäyttöliittymää käytetään paikallisesti robotin koneessa.

3.1 Ubuntu

Etäkoneisiin asennettiin Ubuntun versio 14.04 LTS (Long Term Support), koska tämä oli sillä hetkellä viimeisin stabiili LTS-versio. Robotin päällä käytettyihin koneisiin asennettiin Lubuntu 14.04 LTS, joka on Ubuntun kevennetty versio.

Asennuksessa /etc/hosts -tiedostoon tulee automaattisesti rivi 127.0.1.1 hostname, koska jotkin ohjelmat vaativat, että koneen nimi (host name) on aina voitava liittää johonkin IP-osoitteeseen. Jos koneelle annetaan kiinteä IP-osoite, on se asetettava tämän tilalle. [83, Sec. 10.4], [84, Sec. 5.1.1]

Huomattiin, että Ubuntun päivitysten asennusta ei kannata hyväksyä heti niiden ollessa tarjolla, jotta niissä olevat virheet tulevat korjatuksi ennen asennusta. Yhteensopimattomien päivitysten vuoksi aiheutui monesti harmia. Turhauttavimpana mainittakoon Network Managerin rampauttanut päivitys, jonka seurauksena koneella ei päässyt ollenkaan verkkoon. Tällöin oli selvitettävä ja noudettava tarvittavat komponentit verkosta toisella koneella ja asennettava ne sitten muistitikulta uusimpien versioiden päälle, ennen kuin kone saatiin taas verkkoon.

3.2 ROS

Tässä käsitellään lyhyesti ROSin peruskäyttöönotto ja työhakemistojen konfiguroiminen aikaansaatuun ROS-ympäristöön. Työhakemistoja tarvitaan esimerkiksi silloin, kun muutetaan olemassa olevaa ROS-toteutusta. Tällöin ne ovat hyvä keino pitää kehitysversiot erillään alkuperäisistä. Työhakemistojen käyttö on suositeltavaa myös omien pakettien kanssa työskenneltäessä [85].

3.2.1 ROSin käyttöönotto

ROSin asennukseen Turtlebotia varten löytyi hyvä sivusto [86], jonka ohjeiden mukaan asennus sujui lähes ongelmitta. Ohjeen mukaisesti robotin tietokoneeseen asennettiin ROSin Indigo Igloo, joka oli tuolloin viimeisin LTS-versio (Long Term Support).

Sivustolla annetaan muun muassa ohje Turtlebot-spesifien pakettien asennukseen:”Follow the TurtleBot installation instructions by copying every line below ’Source Installation’ into a terminal.” Viitatulta Turtlebot installation instructions -sivulta [87] on kuitenkin suoritettava lisäksi Ubuntu Package Install -osion alla olevat asennukset, jotta kaikki tarvittavat paketit tulevat mukaan. Muutoin asennus sujui ohjeen mukaisesti.

3.2.2 Työhakemistot

ROSissa käytetään pakettien kehitysvaiheessa työhakemistoja, kuten jo aiemmin todettiin (3.2). Työhakemiston sisältämät paketit voivat korvata varsinaisen ROS-asennuksen sisältämät, jolloin niistä voidaan tehdä omia versioita. Yhteen hakemistoon voi lisätä useampia paketteja, mutta voidaan käyttää myös useampia ketjutettuja työhakemistoja, jolloin voidaan vapaammin valita, mitkä paketit ovat kulloinkin käytössä. Ketjutus tapahtuu hyväksi käyttäen kunkin hakemiston setup.bash -tiedostoja. Työhakemistoon voi lisätä myös omia paketteja, jolloin niitä voidaan käyttää ROS-ympäristössä.

Alkuvaiheessa, kun navigointiin liittyviä tiedostoja tutkittiin ja muuteltiin, piti sitä varten tehdä Turtlebotin päällä olevaan X201:een uusi navigation_ws -työhakemisto. Siihen sisällytettiin kaikki navigointipaketit (navigation stack, 2.1.7); operaation kulku perustuu lähteeseen [88].

Ensin luodaan tyhjä työhakemisto ROS-asennuksessa muodostetun työhakemistoketjun jatkoksi:

```
// luodaan työhakemiston lähdekoodihakemisto
mkdir -p ~/navigation_ws/src
// siirrytään luotuun hakemistoon
cd ~/navigation_ws/src
// alustetaan se
catkin_init_workspace
// palataan työhakemiston juureen
cd ~/navigation_ws
//ajetaan vanhan työhakemistoketjun viimeisen työhakemiston setup-tiedosto
source ~/turtlebot/devel/setup.bash
// muodostetaan loppu hakemistorakenne ja luodaan mm. setup.bash-tiedosto
catkin_make
```

Seuraavaksi lisätään navigation stack tähän työhakemistoon:

```
// siirrytään lähdekoodihakemistoon
cd ~/navigation_ws/src
// luodaan alustava .rosinstall-tiedosto lähdekoodihakemistoon
wstool init
// talletetaan paikallinen (tässä stackin) nimi ja latausosoite .rosinstall-tiedostoon
wstool set navigation --git git://github.com/ros-planning/navigation.git
// päivitetään hakemisto, eli ladataan tiedostot
wstool update
// palataan työhakemiston juureen
```

```
cd ~/navigation_ws
// ajetaan make kaikille työhakemiston paketeille
catkin_make
// ajetaan uuden työhakemiston setup
source ~/navigation_ws/devel/setup.bash
```

Työhakemiston uusi setup.bash on asetettava sourceksi ~/.bashrc:hen. Vanha source kannattaa poistamisen sijaan laittaa kommentteihin, mikä helpottaa ketjun muuttelemista tarvittaessa myöhemmin.

Ajetaan vielä rospack profile, jotta tehdyt muutokset päivittyvät varmasti ROS-ympäristöön:

```
rospack profile
```

Viimeisen setup.bash -tiedoston ajamisen kautta ko. työhakemiston polku lisätään ympäristömuuttujaan ROS_PACKAGE_PATH aiemmin määriteltyjen lisäksi. Tulostetaan X201:n ROS-polut, kun käytössä on ROSin vakiopolkujen lisäksi työhakemistot rocon, kobuki, turtlebot ja navigation_ws:

```
echo $ROS_PACKAGE_PATH

/home/user/navigation_ws/src:/home/user/
turtlebot/src: /home/user/kobuki/src: /home/user/rocon/
src:/opt/ros/indigo/share:/opt/ros/indigo/stacks
```

Edellä olevassa kaksi viimeistä ovat ROSin vakiopolut, navigation_ws on edellä lisätty työhakemisto ja loput Turtlebot-spesifien asennusten yhteydessä muodostuneita työhakemistoja. Huomattavaa on, että hakemistopolkulista kasvaa alkupäästään aina uuden työhakemiston lisäämisen yhteydessä. Suoritustiedostoja haetaan polkulistan alusta lähtien, ja oikean nimisen suoritustiedoston löytyessä se suoritetaan. Listassa ensin oleva polku siis ohittaa jäljempänä tulevan.

Polkujen oikeellisuutta on helppo testata paitsi tulostamalla ROS_PACKAGE_PATH, myös komennolla roscd, joka on ROS-spesifi paketin etsivä hakemistonvaihtokomento. Esimerkiksi voidaan kokeilla, onko move_base paketti korvattu työhakemistossa olevalla:

```
user@Turtle:~$ roscd move_base

user@Turtle:~/navigation_ws/src/navigation/move_base$
```

Nähdään, että kyseinen paketti löytyy työhakemistosta navigation_ws, eli alkuperäinen paketti on korvattu toisella versiolla. Vertailun vuoksi sama komento paketille, jota ei ole korvattu työhakemistoissa olevalla:

```
user@Turtle:~$ roscd rviz

user@Turtle:/opt/ros/indigo/share/rviz$
```

Nähdään, että paketti löytyy hakemistosta /opt/ros/indigo/share. Tämä on hakemisto, jonka alta löytyvät kaikki alkuperäiset ROS-paketit.

3.3 Turtlebot

Ensimmäinen Turtlebot oli saatu laboratorioon valmiiksi koottuna. Siinä oli mukana myös minikannettava, jossa oli vanhempi ROS-versio Hydro Medusa valmiiksi asennettuna. Minikannettava oli kuitenkin rikkoutunut jo ennen työn aloittamista ja se jouduttiin korvaamaan Lenovo X201:llä. Se on tehokkaampi kone, mutta kuluttaa myös enemmän virtaa.

Robotissa on kannettavan lataamiseen soveltuva 19 V/ 2.1 A liitäntä, joka on käytettävissä vain robotin ollessa latauksessa [64, p. 4]. Tarkoitus on, että samalla kun robottia ladataan, myös kannettava tietokone latautuu. Aluksi Lenovo X201:lle piti saada robotin alustan liittimeen sopiva latausjohto. Löytämästämme vanhasta laturista katkaistiin toisiopuolen johdin, joka yhdistettiin robotin mukana tulleeseen, sen alustaan sopivaan irtoliittimeen.

Huomattavaa Lenovon latausjohdossa oli se, että siinä on tietokoneen päässä kolme kontaktia normaalin kahden sijaan, mikä aiheutti aluksi pientä ihmetystä. Liittimessä on pyöreä sylinteri, jonka ulkopinnalla on mitatun perusteella maa ja sisäpinnalla positiivinen latausjännite.

Kolmantena liittimessä on em. sylinterin keskellä pinni, joka pienen selvittelyn jälkeen osoittautui kytketyksi laturimallikohtaisen resistanssin läpi (taulukko 9) maahan. Resistanssin suuruus kertoo koneelle, millainen laturi sitä lataa. Tällöin se osaa muun muassa rajoittaa virrankulutusta, jos sitä ladataan heikkotehoisemmalla laturilla, kuin olisi tarpeen. [89]

Taulukko 9. *Laturikohtaiset resistanssit, perustuu lähteeseen* [89]

laturin teho	**resistanssi**
65 W	10 kΩ
90 W	ei kytketty
135 W	0 Ω
170 W	1.5 kΩ

Mitattaessa keskipinnin ja maan väliseksi resistanssiksi todettiin 10 kΩ, joka vastaa siis 65 W laturia. Tämä on käytettävissä olevista arvoista paras meille, kun Turtlebotin antama maksimi latausvirta on vain 2.1 A / 19 V (≈ 40 W). Koneen mukana toimitettu laturi on tyyppikilpensä mukaan teholtaan 90 W ja kykenee antamaan 4.5 A latausvirran 20 V jännitteellä. Tämä aiheutti epäilyksiä Turtlebotin tarjoaman latausliitännän riittävyydestä.

Latausliitännän riittävyyttä testattiin muun muassa jättämällä kone 16 tunniksi lataukseen sen ollessa päällä, mikä on normaali käyttötilanne. Tänä aikana akun kapasiteetti kasvoi 52 -> 57 %, eli vain 5%. Käytännössä tämä ei alkuunkaan riitä jatkuvaan käyttöön.

3.4 Hadoop

Yleensä HDFS:ää käytetään MapReducen yhteydessä, mutta tässä työssä sitä käytetään ainoastaan robottien karttatiedon tallennuspaikkana. Meidän tapauksessamme clientina toimii robotin kannettava tietokone, jolla haetaan tiedostoja HDFS:tä ja/tai tallennetaan niitä sinne. Operointi tapahtuu terminaalin kautta.

Useamman koneen Hadoop-klusteri on helpointa toteuttaa asentamalla tarvittaviin koneisiin ensin yhden koneen klusteri ja muodostamalla niistä sitten monen koneen kokoonpano [90]. Alkuun perustettiin kolme yhden koneen Hadoop-klusteria (single node cluster) perustuen lähteisiin [90], [91] ja [92]. Ohjeet ovat Hadoop versioille 1.0.3, 2.6.0 ja 2.7.1, mutta soveltuivat myös versiolle 2.7.2, joka OpenCRP-ympäristöön asennettiin.

Asennus sujui ongelmitta, ohjeita hieman soveltaen. OpenJDK:n sijaan koneisiin asennettiin Oraclen JDK 7, kuten Apachen sivuilla suositeltiin [93]. Yhden koneen klusterin asennuksen tarkempi kulku käy ilmi liitteestä A, johon on kirjattu suoritetut toimenpiteet ja kommentoitu parametreja tarvittavilta osin.

Seuraavaksi konfiguroitiin kahden koneen Hadoop-klusteri ja yksi client perustuen lähteisiin [90] ja [94]. Hadoop-master, -slave ja -client koneille annettiin nimiksi vastaavasti Hdmaster, Hdslave ja Turtle. Näiden hosts-tiedostoihin lisättiin muiden koneiden nimet ja IP-osoitteet. Tällöin kaikkia Hadoopin konfigurointitiedostoja ei tarvitse muuttaa, jos koneiden kiinteitä IP-osoitteita joudutaan muuttamaan.

Myös kahden koneen klusterin ja siihen liittyvän clientin konfigurointiaskeleet kirjattiin. Edellä mainittujen lähteiden ohjeita jouduttiin soveltamaan tilanteen mukaan, ennen kuin toimiva lopputulos saatiin aikaiseksi. Konfiguroinnin kulku on kuvattu liitteessä B.

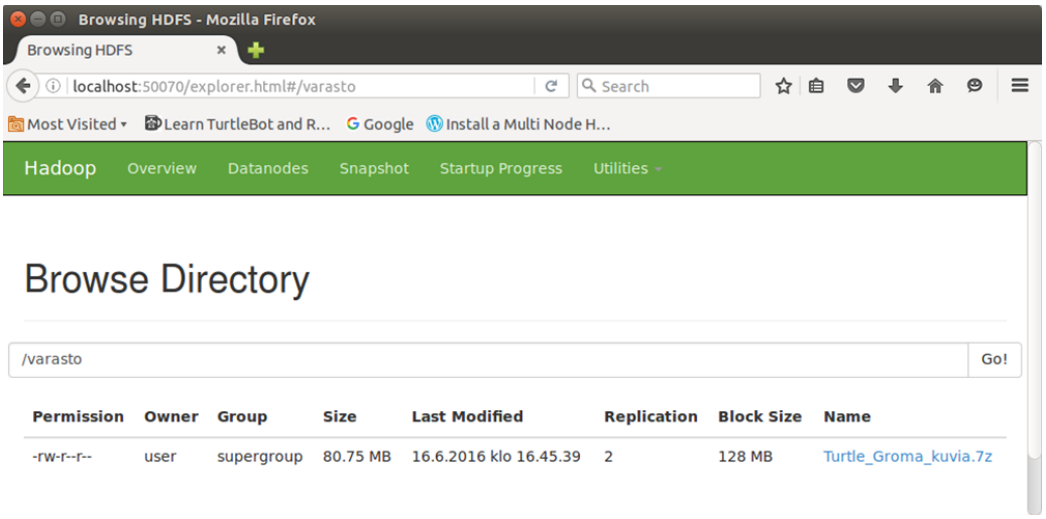
Lopuksi todettiin koko Hadoop-konfiguraation toiminta. Käynnistettiin HDFS ja MapReduce Hdmasterin terminaali-istunnossa:

```
start-dfs.sh        // HDFS:n käynnistys
start-yarn.sh       // MapReducen käynnistys
```

Luotiin clientilta (Turtle) tallennushakemisto nimellä varasto ja siirrettiin sinne suuri tiedosto (>80 MiB):

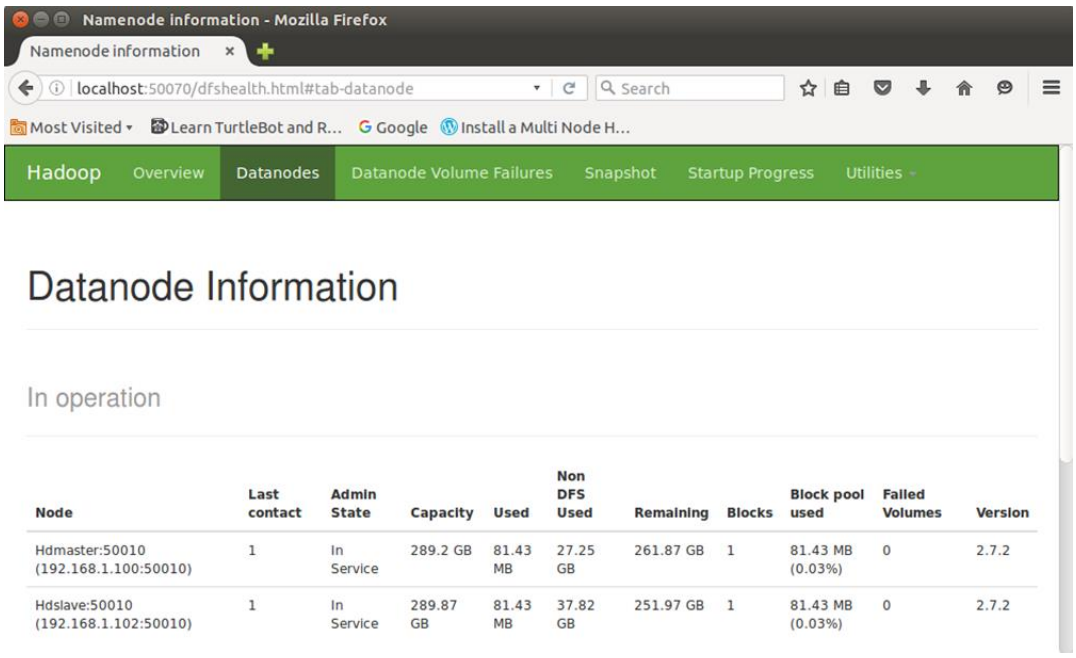
```
hadoop fs -mkdir /varasto

hadoop fs -put Turtle_Groma_kuvia.7z /varasto
```

Katsottiin hakemistosisältö selaimella (kuva 12):



Kuva 12. *HDFS:n hakemistosisältö*

Siirretty tiedosto näkyi hakemistossa ja sen replikointitekijänä (Replication) näkyi kaksi, kuten pitikin. Tiedoston tuli siis olla tallennettuna yhdessä blokissa sekä Hdslave- että Hdmaster-koneilla, koska blokin koko (Block Size) oli 128 MB. Varmistettiin tämä selaimella (kuva 13):



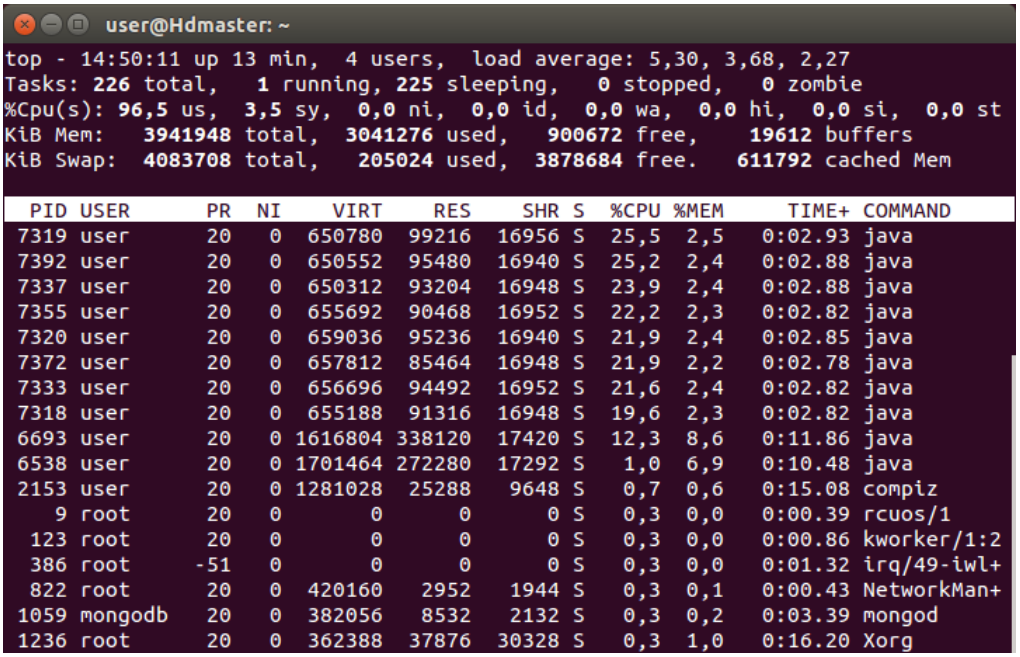
Kuva 13. *HDFS:n datanode-näkymä*

Tiedosto näkyi molemmilla koneilla yhden blokin kokoisena (Blocks), kuten pitikin. Myös tiedoston haun ja poistamisen todettiin toimivan niitä kokeilemalla. HDFS:n osalta konfiguraatio toimi odotusten mukaisesti.

MapReduce:n toiminta todettiin ajamalla clientilta valmis esimerkkiskripti:

```
yarn jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar pi 30 100
```

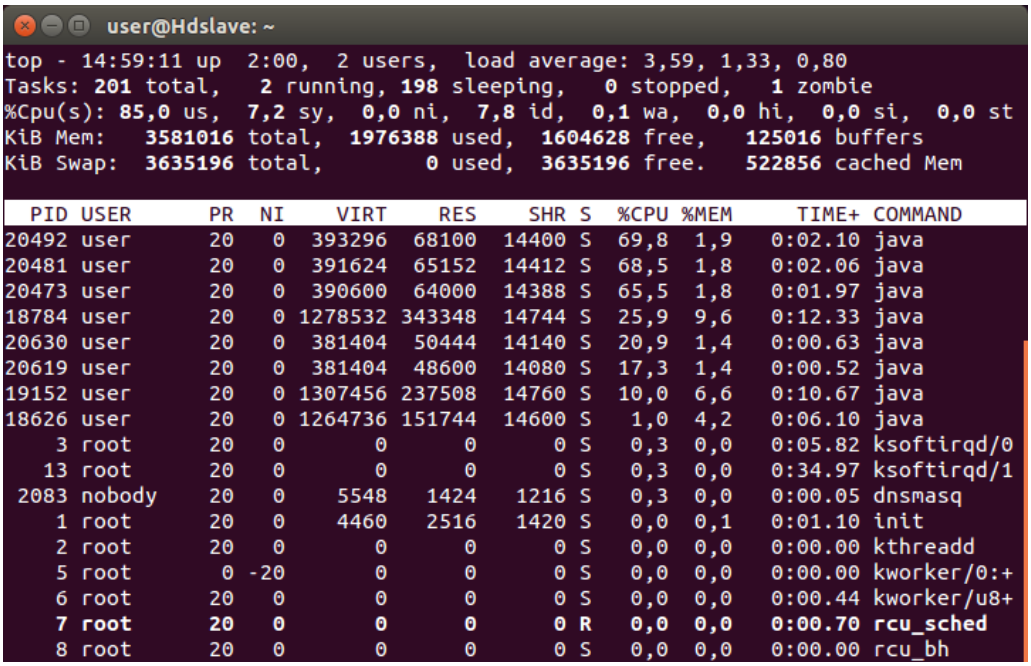
Skriptin suorituksen aikana seurattiin Hdmasterin ja Hdslaven prosessikuormia, sekä otettiin niistä kuvakaappaukset (kuvat 14 ja 15). Kuvissa käyttäjäprosessien kokonaisajankäyttö on näytön yläosan %Cpu(s) -rivin ensimmäinen lukema (us = user), muilla näytön yläosan lukemilla ei tässä tapauksessa ole merkitystä. Kuvakaappaukset otettiin muutaman sekunnin välein. Niissä näkyvä usean minuutin aikaero johtuu siitä, etteivät koneiden kellot olleet synkronissa keskenään.



Kuva 14. *Hdmaster, prosessikuorma*

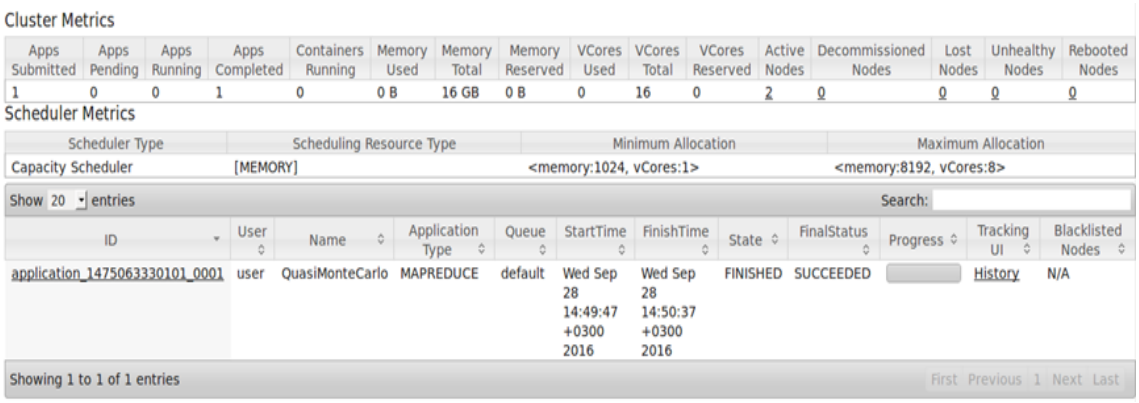
Ennen skriptin käynnistämistä molempien koneiden käyttäjäprosessien kuormat olivat 2 %:n molemmin puolin. Tässä MapReducen jakamia Java-prosesseja (COMMAND) näkyi masterissa ajossa useita ja prosessorikuorma oli noussut lähelle täyttä kapasiteettia ollen 96.5%.

Kuvia tulkitessa on muistettava, että kuvan alaosan *prosessilistassa* näkyvät %CPU-lukemat tarkoittavat suhteellista ajankäyttöä ydintä, tai hypersäikeistystä [95] käytettäessä virtuaaliydintä kohti. Niiden summa voi näin ollen kasvaa yli 100 %:n. Molemmissa koneissa on kaksi ydintä, mutta Hdslavessa hypersäikeistyksen myötä 4 virtuaaliydintä.



Kuva 15. *Hdslave, prosessikuorma*

Sama tilanne nähtiin slave-koneessa, jossa oli myös ajossa useita Java-prosesseja. Laskenta jaettiin siis molemmille klusterin koneille odotusten mukaisesti. Laskennan tiedot todettiin myös Hadoopin selainkäyttöliittymän kautta (kuva 16).



Kuva 16. *Osa Hadoopin web-käyttöliittymän All Applications -näkymää*

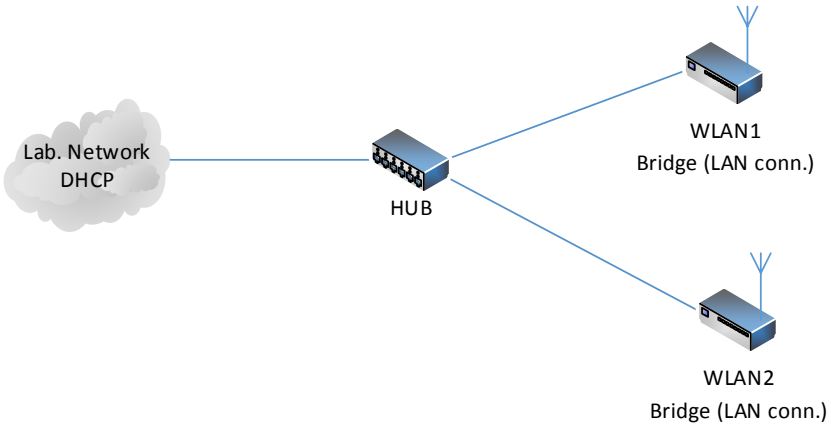
Kuvasta nähdään, että laskentatehtävä suoritettiin onnistuneesti 50 s kuluessa (Finish Time, Start Time). Laskennan lopputulos oli todettavissa clientin terminaalissa, jossa laskennan etenemistä saattoi sen aikana seurata map- ja reduce-taskien suoritettujen prosenttiosuuksien muodossa. HDFS:n lisäksi MapReducen voitiin todeta toimivan asianmukaisesti ja olevan käyttövalmis, mikäli sitä projektin jatkuessa tullaan tarvitsemaan.

3.5 Robottien tiedonsiirto

Robotin ja etäkoneiden välinen tiedonsiirto hoidetaan OpenCRP:ssä WLANin välityksellä. TTY:n Porin laitoksen tietoliikennelaboratoriossa ja sen ympäristössä on paljon

WLAN-tukiasemia, joten heti ensimmäiseksi etsittiin OpenCRP:n tukiasemalle kanava, jolla saavutettaisiin paras yhteyden laatu. Tähän käytettiin ilmaista inSSIDer-ohjelmaa [96], joka pisteyttää kanavat paremmuusjärjestykseen.

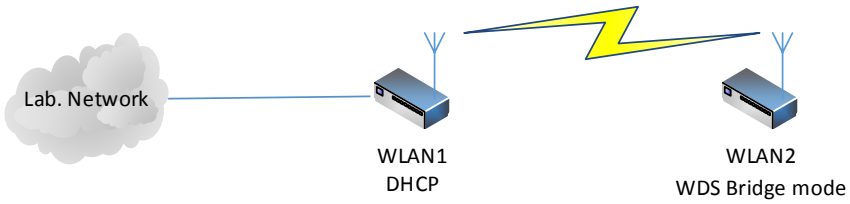
Koska yhden tukiaseman peitto on varsin rajallinen, kokeiltiin peiton kasvattamiseksi kahta erilaista konfiguraatiota. Ensin kytkettiin HUB laboratorion verkkoon ja siihen kaksi tukiasemaa siltana, eli jatkettiin laboratorion verkkoa WLAN-tukiasemilla (kuva 17).



Kuva 17. *HUB ja tukiasemat siltana*

ROS-ympäristössä muiden koneiden pitää tuntea ROS-master -koneen osoite ja niin ikään Hadoop klusterissa koneiden on tunnettava toistensa osoitteet. Koska haluttiin pitää konfiguraatio sellaisena, että tullaan toimeen myös ilman DNS-palvelinta, annettiin koneille kiinteät IP-osoitteet. Tässä robotti ja WLANiin kytketyt etäkoneet saavat IP-osoitteet laboratorion verkon DHCP-palvelimelta, joten niiden kiinteät IP:t otettiin sen osoiteavaruudesta.

Toinen kokeiltu tapa tukiasemapeiton laajentamiseksi oli kytkeä toinen tukiasema laboratorion verkkoon ja toinen WDS-moodissa (Wireless Data Distribution), eli kloonattuna langattomana yhteyspisteenä (kuva 18). Laboratorion verkkoa tarvitaan tässä vain internet-yhteyden saamiseksi koneille. Tämä ei ole tarpeen ROSin kannalta, vaan kytkettiin vain helpottamaan työskentelyä, kun samoilta koneilta pääsi myös internetiin.



Kuva 18. *WDS-silta*

Tässä robotti ja etäkoneet saavat IP-osoitteet WLAN 1:ssä ajettavalta DHCP-palvelimelta, joten niiden kiinteät IP-osoitteet asetettiin sen osoiteavaruudesta. Huonona puolena tässä kytkennässä on ilmatien kapasiteetin tuhlaaminen tukiasemien väliseen liikenteeseen.

Tukiaseman ”sivistynyttä” vaihtamista varten on olemassa käyttäjäparametri ”roaming aggressiveness”, jolla voidaan määrittää signaalitaso, jolla verkon käyttäjä alkaa etsiä parempaa tukiasemaa. Valitettavasti käytössämme olleiden kannettavien WLAN-kortit eivät tukeneet tätä parametria. Tästä seurasi se, että uutta tukiasemaa alettiin etsiä vasta, kun yhteys meni käytännössä poikki. Molemmilla konfiguraatioilla tehtiin roaming-mittauksia verkon vaihtamisen nopeuden selvittämiseksi, nämä käsitellään myöhemmin muiden mittausten yhteydessä (3.7).

3.6 Testikäyttöliittymä

Ohjelmiston kehitysprosessi voi konservatiivisimmillaan olla suhteellisen raskas ja jäykkä tiukasti seuraavine työvaiheineen ja vaihetuotteineen. Prosessi on syytä sovittaa tuotteeseen nähden tarkoituksenmukaisesti tällaisen suunnitelmavetoisen (plan driven) ja kevyemmän ketterän (agile) menetelmän välimaastossa. [97, pp. 62–64]

Testikäyttöliittymän toteutuksen ensisijaisena tavoitteena oli selvittää, tulevan kehityksen pohjaksi, miten ROS-node ja sen graafinen käyttöliittymä liitetään toisiinsa. Toisena tavoitteena oli saada aikaan mahdollisimman helppokäyttöinen testikäyttöliittymä, jonka avulla robotin kulkua voidaan hallita koekäytössä.

Aikataulullisista ja edellä mainituista syistä valittiin lähestymistapa, jossa kehitysprosessi pidettiin erittäin kevyenä ja vapaamuotoisena. Dokumentointi rajoitettiin vaatimus- ja testitapauslistaan, graafisen liittymän sijoittelusuunnitelmaan sekä koodin selkeään kommentointiin. Nimeksi käyttöliittymä sai T*urtleohj* (Turtlebot-ohjaus). Sen jatkokehitys tiettyyn kohdeympäristöön vaatii tapauskohtaisen lähestymistavan yhteistyössä asianomaisten sidosryhmien kanssa.

3.6.1 Vaatimukset

Käyttöliittymää käytetään robotin päällä olevassa koneessa ja/tai siihen WLANilla yhdistetyssä etäkoneessa. Navigoinnin mahdollistavat ROS-käynnistystiedostot (bringup_minimal.launch ja amcl_demo.launch) ajetaan omissa terminaaleissaan käsin ja tämän jälkeen käynnistetään turtleohj-käyttöliittymä. Koordinaatit eri pisteille saadaan erikseen ROSin rviz-työkalulla [98], jossa ne saadaan näkyviin halutuissa karttapisteissä. Master URI:lle ja host IP:lle ei tarvitse tehdä muototarkastuksia.

1. ROS-master URI ja -host IP ovat käyttäjän konfiguroitavissa ohjelman käynnistyksessä

2. edellä mainitut valittavissa myös luettavaksi järjestelmämuuttujista
3. jos ROS-masteriin ei saada yhteyttä, siitä on kerrottava käyttäjälle
4. valittavissa on vaihtoehto, jolla yhteys käynnistetään automaattisesti seuraavassa turtleohj-käynnistyksessä
5. asetukset säilyvät ohjelman uudelleenkäynnistyksessä
6. käytettävissä on muutama navigointipainonappi, joita painamalla voi valita kohteen, johon robotti ajaa (napin painalluksesta parin sekunnin viive robotin lähtöön)
7. painonapit ovat käyttäjän nimettävissä
8. painonappia vastaavat karttakoordinaatit ovat käyttäjän asetettavissa
9. navigointi- ja pysäytysnappien painaminen on estetty, kun masteriin ei ole yhteyttä
10. käynnistysnapin painaminen on estetty, kun masteriin on yhteys
11. navigointinapit, joita ei ole nimetty, eivät ole käytettävissä
12. ohjelma sammuu painonapista painamalla
13. robotin voi pysäyttää painonapista painamalla, kun se on liikkeellä
14. Käyttöliittymä voi olla käynnissä robotin päällä olevassa koneessa ja etäkoneessa samanaikaisesti

3.6.2 Sijoittelusuunnitelma

Käyttöliittymä suunniteltiin MS Officen Visiolla käyttäen Controls-, Dialogs- ja Toolbars-muotokokoelmia. Päämääränä oli tehdä käyttöliittymästä mahdollisimman yksinkertainen ja helppokäyttöinen.

Pop-up -ikkunoiden määrä haluttiin rajoittaa ilmoitusten ja ohjelmatietojen esittämiseen. Tästä syystä taulukko painonappien nimien ja arvojen määritystä varten tehtiin navigoinnin kohdepainikkeiden kanssa päällekkäin. Näistä toinen on näkyvissä aina kulloisenkin käyttötilanteen vaatimalla tavalla. Suunnitelma on liitteessä C.

3.6.3 ROS-QT -kehitysympäristö

Ohjelmointityötä varten asennettiin Qt:n versio 5.5.1 open source -asennuksena kolmeen projektin koneeseen. Ohjelmiston online-asennustiedosto on ladattavissa lähteestä [99]. Asennusta varten oli avattava Qt-tili, jonka avasin omin sähköpostitunnuksin.

Projektiympäristön, jossa ROS ja Qt saadaan sovitettua samaan projektiin, aikaan saaminen oli haaste. Onneksi tähän löytyi ROSista oiva työkalu *catkin_create_qt_pkg*, jonka käytöstä löytyy tietoa lähteestä [100, Indigo]. Sillä voi luoda ”Hello world” -tyyppisen pakettitemplaten, joka on valmis yksinkertainen ROS-Qt -projektipohja. Meille hyödyllisinä komponentteina se sisältää valmiiksi erilliset UI- ja ROS-nodeluokat, sekä masterin URI:n ja hostin IP:n asetukset.

Templatesta sai hyvän pohjan omalle käyttöliittymäohjelmalle, tosin kehitysympäristö ei toiminut sen kanssa aivan kaikilta osin. Esimerkiksi tapahtumakäsittelijöiden runkoja ei saanut luotua objektin kautta automaattisesti, vaan ne piti tehdä käsin sekä cpp- että hpp-tiedostoihin. Kaikkiaan tämä kuitenkin helpotti ohjelmointiprojektin luomista Qt-ympäristöön suuresti.

Pakettia varten luotiin ensin oma työhakemisto, joka linkitettiin konekohtaisesti aiempaan työhakemistoketjuun (ks. 3.2), sitten luotiin Qt-paketti, jolle ajettiin make. Tarvittavat komennot, perustuu lähteisiin [88] ja [100, Indigo]:

```
mkdir -p ~/qt_workspace/src                          // luodaan uusi työhakemisto
cd ~/qt_workspace/src                                // siirrytään sinne
catkin_init_workspace                                // alustetaan se
source ~/navigation_ws/devel/setup.bash              // ws-ketjun edellinen ws
catkin_create_qt_pkg turtleohj                       // luodaan pakettitemplate
sudo apt-get install ros-indigo-qt-build             // tarvittaessa asennetaan qt_build
catkin_make --force-cmake                            // ajetaan make
```

Viimeistä komentoa voi kokeilla myös ilman -force-cmakea. Sen tarkoitus on kertoa catkinille, että valmiiksi käännettyyn ympäristöön on tullut uusi paketti, joka tulee ottaa mukaan käännökseen [101]. Lisäksi oli muistettava lisätä uusi työhakemistoketjun viimeinen lenkki ~/.bashrc:hen ja kommentoida samalla edellinen.

Koska ~/qt_workspace/src -hakemistossa oleva CMakeLists.txt on viittaus varsinaiseen tiedostoon /opt/ros/indigo/catkin/cmake/toplevel.cmake ja Qt IDE yrittää kirjoittaa omaa tiedostoaan samaan hakemistoon sen kanssa, kopioitiin sen tilalle toplevel.cmake.

```
~/qt_workspace/src -hakemistossa:
cp /opt/ros/indigo/share/catkin/cmake/toplevel.cmake CMakeLists.txt
```

Tästä on seurauksena se, että Qt kirjoittaa oman versionsa CmakeLists-tiedostosta (CmakeLists.txt.user) tähän työhakemistoon ROS-perusasennushakemistojen sijaan. Tällöin vältytään myös kirjoittamasta esimerkiksi toisen projektin tiedoston päälle, jos meneillään on useampia ROS-Qt -projekteja.

3.6.4 GitLab-versionhallinta

Versionhallintaa varten asennettiin yhteen laboratorion koneista Git-palvelin käyttämällä GitLabin avoimen lähdekoodin Community Editionia v. 8.10.3. Konfiguraatiossa sivuutettiin sähköpostipalvelimen asetukset, koska meillä ei ole sille tällä hetkellä käyttöä. Uudelleenkonfigurointi on tarvittaessa mahdollinen myöhemminkin.

Palvelimelle luotiin GitLabin web-käyttöliittymän kautta uusi repository nimellä turtleohj, jonka jälkeen sinne tallennettiin koko ~/qt_workspace/src/turtleohj-kansion sisältö (komennot annettava em. kansiossa):

```
// alustetaan hakemisto -> luo paikallisen .git repositoryn
git init
// lisätään kaikki ko. hakemiston tiedostot ja hakemistot indeksiin (cache)
git add .
// lisätään em. hakemistot ja tiedostot repositoryyn
git commit -m "Initial commit"
// luodaan uusi etätallennuspaikka (remote) annetulla osoitteella -> nimeksi origin
git remote add origin http://Git/opencrp/turtleohj.git
// talletetaan tiedostot ko. repositoryyn etätallennuspaikassa origin
git push -f origin master
```

Jotta myös muihin koneisiin saatiin ajantasainen projektiympäristö, luotiin niihin aluksi ROS-Qt -ympäristö (3.6.3). Tämän jälkeen poistettiin niihin templaten luonnissa syntynyt ~/qt_workspace/src/turtleohj -hakemisto ja kloonattiin Gitistä sen tilalle uusi ja ajantasainen ohjelmatiedostorakenne (komento ~/qt_workspace/src -hakemistossa):

```
// kloonataan tiedostot ja tehdään .git-piilohakemisto ko. sijaintiin
git clone http://Git/opencrp/turtleohj.git
```

Tässä tarvittiin käyttäjätunnus ja salasana ko. repositoryyn. Kun tiedostojen haku oli suoritettu, avattiin projekti ja ”luotiin” Qt:ssä Git repository (Tools/Git/Create Repository) turtleohj-hakemistoon. Kun Qt kysyi, haluatko valita toisen hakemiston, kun tämä jo on versionhallinnan alainen Git-hakemisto, vastattiin ”ei”, jotta ohjelma ottaa valmiin Git-versionhallintahakemiston käyttöön. Tämän jälkeen todettiin ympäristön toiminta ajamalla rebuild all (Build/Rebuild All).

Koodimuutokset talletetaan Gitiin tekemällä Qt:ssä committeja paikalliseen repositoryyn (Tools/Git/Local Repository/Commit) ja siirtämällä sieltä muutokset Git-palvelimelle (Tools/Remote Repository/Push). Kun palvelimelle on päivitetty uusi versio, se saadaan haetuksi muille koneille suorittamalla *pull* (Tools/Git/Remote Repository/Pull). Koska koodia muokkasin yksin minä, haaroittamista ei tarvinnut ajatella ollenkaan. Useamman yhtaikaisen haaran hallinnasta ja Gitin periaatteesta ylipäätään löytyy lisätietoa lähteestä [102].

3.6.5 Ohjelmointi

Käyttöliittymä toteutettiin siis valmiin ROS-Qt -projektipohjan (3.6.3) päälle. Sieltä siivottiin kaikki tarpeeton pois ja tehtiin julkaisija sanomalle PoseStamped topiciin

/move_base_simple/goal, johon ko. sanomalla julkaistaan robotille kohdekoordinaatit, joihin sen halutaan liikkuvan. Tämän topicin kautta koordinaatit välitetään robotin liikkumisesta vastaavalle move_base -nodelle. Seuraavassa julkaisijan ja sanoman julkaisun vaatimat koodit:

```
ros::NodeHandle n; //esitellään nodelle kahva, jolla siihen voidaan viitata

// esitellään kohdekoordinaattien julkaisija PoseStamped-sanomalle topiciin
// move_base_simple/goal ja asetetaan sanomapuskurin kooksi 1000
position_publisher =
n.advertise<geometry_msgs::PoseStamped>("move_base_simple/goal", 1000);

// positiosanoman julkaisufunktio
void QNode::sendPose(double x, double y, double w){
   geometry_msgs::PoseStamped pose_msg; // sanoman esittely
   pose_msg.header.stamp = ros::Time::now(); // headerin aikaleiman täyttö
   pose_msg.header.frame_id = "/map"; // headerin frame id:n täyttö
   pose_msg.pose.position.x = x; // kohdepisteen x-koordinaatti
   pose_msg.pose.position.y = y; // kohdepisteen y-koordinaatti
   pose_msg.pose.orientation.w = w; // robotin rintamasuunta kohdepisteessä
   position_publisher.publish(pose_msg); // julkaistaan pose-sanoma
}
```

Lisäksi tehtiin toinen julkaisija topiciin /move_base/cancel, johon julkaistu tyhjä sanoma peruuttaa aiemmin annetut kohteet, eli pysäyttää robotin siihen missä se sanoman saapuessa oli. Seuraavassa tämän sanoman julkaisija ja julkaisufunktio:

```
// esitellään kohdekoordinaattien peruutuksen julkaisija GoalID-sanomalle
// topiciin move_base/cancel ja asetetaan sanomapuskurin kooksi 1000
stop_msg_publisher =
n.advertise<actionlib_msgs::GoalID>("move_base/cancel", 1000);

// pysäytyssanoman julkaisufunktio
void QNode::sendStop(){
   actionlib_msgs::GoalID stop_msg; // sanoman esittely
   // sanoman julkaisu, tyhjä sanoma peruuttaa kaikki kohteet, mukaan
   // lukien käynnissä oleva siirtymä
   stop_msg_publisher.publish(stop_msg);
}
```

Nämä olivat projektipaketin luomisen lisäksi testikäyttöliittymän kannalta tärkeimmät selvitettävät asiat, muutoin liittymän ulkoasu ja toiminnallisuus toteutettiin vaatimuslistan mukaisiksi, eikä niiden ohjelmointia käsitellä tässä sen enempää.

Ohjelma on dynaamisesti linkitetty. Tämä tarkoittaa sitä, että ajettavan tiedoston mukaan ei ole linkitetty sen käyttämiä Qt-kirjastoja. Se käyttää tällöin ROSin asennuksen yhteydessä asentuneita Qt-kirjastoja.

Testikäyttöliittymän lähdekoodit säilytetään ainakin projektin käynnissä oloajan TTY:n Porin laitoksen tietoliikennelaboratoriossa sijaitsevalla palvelimella (3.6.4). On mahdollista, että koodi julkaistaan GitHubissa myöhemmin.

3.6.6 Testit

Vaatimusten perusteella tehtiin suoraan testipöytäkirja, joka sisältää myös testitapausten hyväksymiskriteerit. Testejä varten laboratoriosta ajettiin robotilla osittainen kartta, joka tallennettiin suoraan sen tietokoneelle.

Testauksessa robotilla ajettiin laboratoriossa muutaman koordinaattipisteen väliä. Testitapauksissa asetettavien koordinaattien lukeminen kartalta on ohjeistettu testipöytäkirjan alussa. Testipöytäkirja tuloksineen on liitteessä D.

3.6.7 Lopputulos

Käyttöliittymän ulkoasusta on kuvakaappaukset liitteessä E. Qt:n versionumero ikkunassa ”About Qt” poikkeaa toteutukseen käytetystä Qt-versiosta, koska projektipaketti rakennettiin ROSin catkin_create_qt_pkg -työkalulla [100], joka luo siitä Qt4-paketin.

Testikäyttöliittymän toimintaympäristön havainnollistamiseksi muodostettiin ROSin rqt_graph -työkalulla laskentagraafi ajonaikaisesta järjestelmästä, kun Turtlebotia käytetään navigointiin testikäyttöliittymällä (liite F). Liitteen kuvaan on suodatettu vain käytössä olevat debuggaukseen (esim. konsolitulostukseen) liittymättömät nodet. Testikäyttöliittymä-node on merkitty graafiin punaisella ympyrällä. Topic /move_base/cancel näkyy kaaviossa nimellä /move_base/action_topics, koska rqt_graph yhdistää useampia topiceja yhteen action_topics -instanssiin.

Käyttöliittymä on havaittu paitsi testein, myös käytännössä hyvin toimivaksi ja vaatimukset täyttäväksi. Sen pohjalta on helppo lähteä kehittämään lisäominaisuuksia. On kuitenkin muistettava, että kyseessä on vain testikäyttöliittymä ja kuten aiemmin on jo todettu (3.6), käyttöliittymän kehitys tiettyyn ympäristöön on tehtävä pitkälti asiakkaan ehdoilla ja voi johtaa tapauskohtaisesti hyvin erilaiseen lopputulokseen. Tästä huolimatta tässä kerättyä tietoa tullaan varmasti tarvitsemaan myös jatkokehityksessä.

3.7 Havaintoja ja mittauksia

Ympäristöä ja sen osia testattiin ongelmakohtien selvittämiseksi. Testaus piti sisällään robotin kartoituksen ja navigoinnin kokeilua käytännössä, sekä langattoman verkon tiedonsiirron luotettavuuden mittaamista tukiasemalta toiselle siirryttäessä.

Robotin toiminnallisuuden ja tiedonsiirron lisäksi testattiin myös 3D-sensoreiden toimintaa. Niiden ominaisuuksilla on robotin suorituskykyyn suuri vaikutus, sillä sekä kartoitus että navigointi perustuvat meidän kokoonpanossamme suurelta osin 3D-sensorin mittausdataan.

3.7.1 3D-Sensorit

Molemmat käyttämämme sensorit, sekä Asuksen Action Pro Live että Microsoftin Kinect for Xbox, käyttävät etäisyyksien mittaamiseen laserpistekuviota joka projisoidaan mitattavaan ympäristöön. Kuvio luetaan IR-kameralla ja on sitä kautta nähtävissä, mutta kumpikaan yhtiö ei ole julkistanut menetelmän tarkkaa kuvausta ja saatavilla olevat kuvaukset perustuvat lähinnä takaisinmallinnukseen ja asiaan liittyviin patentteihin. Kuviosta käytetään oletuksena 10 pisteen paksuinen siivu, jonka vertikaalisuunnassa lähimmät mittaustulokset projisoidaan samaan tasoon. Näin saadaan aikaan laserskannausta emuloiva lasermittausviiva.

3D-sensorien merkittävimmät ominaisuudet OpenCRP:ssä ovat minimi- ja maksimimittausetäisyys. Jos minimietäisyys on suuri, lähellä olevat kohteet jäävät havaitsematta. Jos taas maksimietäisyys on pieni, paikannus kärsii AMCL-algoritmin saadessa vähemmän mittausdataa, jota verrata pohjakarttaan. Molemmat etäisyydet mitattiin asettamalla robotti kohtisuoraan mittauspintaa vastaan. Mittauksissa käytettiin rvizin robottinäkymää, johon oli valittu näkyviin laserkeilaus.

Minimimittausetäisyys mitattiin etäisyytenä sensorin etupinnasta kolmeen erilaiseen seinäpintaan. Haarukoimalla etsittiin aina pienin etäisyys, jolla lasermittausviiva oli vielä yhtenäinen keskellä kuvaa, sillä siitä mittaustulos katoaa ensimmäiseksi etäisyyden pienentyessä.

Etäisyydet mitattiin mittanauhalla ja tulokset pyöristettiin lähimpään puoleen senttimetriin, sillä yhtenäisen viivan määrittäminen väreilevästä laserviivasta ei ollut täysin yksiselitteistä. Tulokset ovat taulukossa 10.

Taulukko 10. *Minimimittausetäisyydet eri materiaaleille*

Seinän laatu	**Kinect Xbox 360/min. et. cm**	**Xtion Pro Live/min. et. cm**
Vaalea lasikuitutapetti	54.5	55.5
Harmaa peltikaappi	60.5	63.5
Valkoinen ovi	65.0	77.5
Punainen tiiliseinä	50.0 (tiilen pinnasta)	49.0 (tiilen pinnasta)

Mattapintaisesta tiiliseinästä saatiin molemmille sensoreille parhaat tulokset. Näyttäisi siltä, että himmeistä pinnoista saadaan parempia tuloksia, kuin kiiltävistä. Syytä tähän on vaikea arvioida tarkemmin, koska sensorien tarkka toimintaperiaate ei ole tiedossa.

Mitattu minimimittausetäisyys oli Kinectillä 0.50 – 0.65 m, joka on parempi, kuin valmistajan sivuillaan julkaisema 0.8 m [66]. Lähteessä [69] mainitaan OpenKinect -ohjelmiston rajoittavan minimimittausetäisyyden 0.5 m:iin, mikä oli myös pienin tässä tulokseksi saatu mitattausetäisyys.

Xtion Prolle valmistaja ilmoittaa minimimittausetäisyydeksi myös 0.8 m [71]. Mitatessamme sensoreille haastavimmaksi osoittautuneeseen valkoiseen oven pintaan saimme minimimittausetäisyydeksi 0.77 m, joka on lähellä ilmoitettua. Sensori pystyy kuitenkin mittauspinnasta riippuen havaitsemaan myös tätä lyhyempiä etäisyyksiä, kuten punaisesta tiilipinnasta tehty 0.49 m mittaustulos osoittaa.

Maksimimittausetäisyys mitattiin vain kahdenlaiseen pintaan, koska tarpeeksi pitkiä mittauspaikkoja muihin ei löytynyt. Mittauksessa robotti nostettiin pyörillä kulkevan pöydän päälle ja etäisyydet haarukoitiin pöytää liikuttamalla. Kullekin pinnalle mitattiin kolme rajaetäisyyttä:

1. laserviiva on vielä staattinen koko matkalta (pieniä rakoja näkyy luonnostaan ilmaantuvan etäisyyden kasvaessa molemmissa sensoreissa)
2. viivassa on vielä erotettavissa staattisia osuuksia, eli niistä saataisiin mittausdataa
3. mittaustulokset katoavat käytännössä koko mittauskohdan leveydeltä (satunnaiset pilkahdukset jätetty huomiotta)

Etäisyydet mitattiin Bosch PLR 15 -lasermittarilla, jonka mittaustarkkuus on ± 3.0 mm [103]. Tulokset (taulukko 11) merkittiin kuitenkin 10 cm:n tarkkuudella, koska tulkinta oli tässä vielä minimietäisyysmittaustakin vaikeampi tehdä yksiselitteisesti.

Taulukko 11. *Maksimimittausetäisyydet eri materiaaleille*

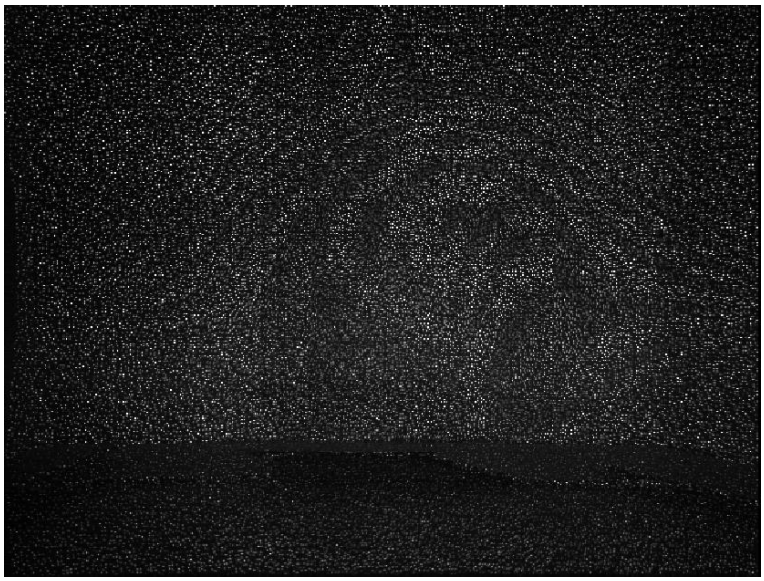
Seinän laatu	**Kinect Xbox 360/min. et. m**	**Xtion Pro Live/min. et. m**
Vaalea lasikuitutapetti	6.1, 9.3, 9.7	7.1, 9.0, 10.1
Valkoinen ovi	5.7, 7.9, 8.3	5.8, 7.8, 10.0

Molemmille sensoreille osoittautui tässäkin mittauksessa haastavammaksi mitattavaksi valkoinen ovi. Kinectissä havaittiin epätasaisuutta siten, että viivan staattiset osat katosivat joillakin etäisyyksillä palatakseen etäisyyden kasvaessa. Tällainen kohta oli lasikuitutapettiin mitattaessa 8.6 m:n ja oveen mitattaessa 7.0 m:n ympäristössä. Xtion Prossa ei tällaisia vyöhykkeitä ollut havaittavissa, vaan käytös muuttui johdonmukaiseti etäisyyden mukaan.

Kinectille valmistaja ilmoittaa maksimimittausetäisyydeksi 4 m [66]. Lasermittausviiva oli kuitenkin oveen mitatessa koko matkaltaan staattinen vielä 5.7 m etäisyydeltä, joka on selvästi ilmoitettua parempi.

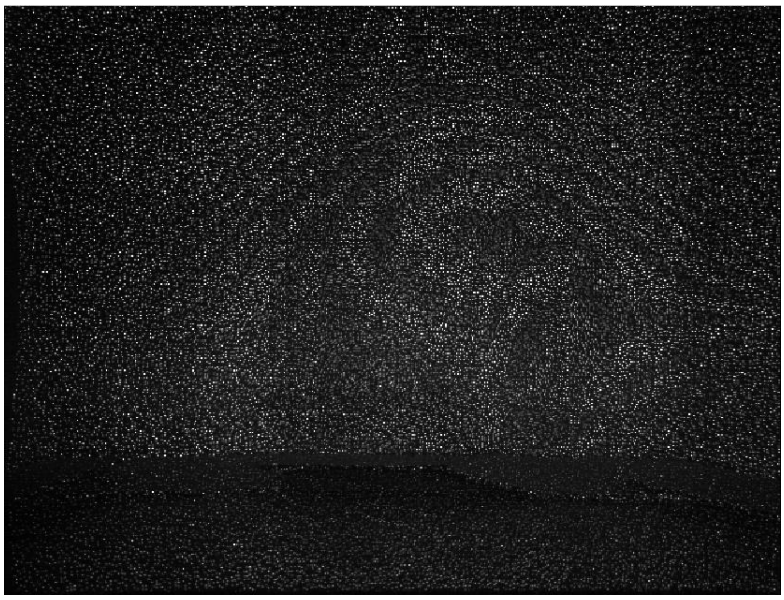
Xtion Prolle valmistaja ilmoittaa maksimimittausetäisyydeksi 3.5 m [71]. Kinectin tapaan myös Xtion Prolle saatiin ilmoitettua selvästi parempi tulos koko lasermittausviivan ollessa stabiili vielä 5.8 m matkalla.

Koska sensorien mittaus tapahtuu IR-projektorin heijastaman laserpistekuvion perusteella, haluttiin selvittää myös *valaistusolosuhteiden vaikutusta* niiden mittausnäkymään. Tätä varten otettiin kuvia Kinectin IR-kameralla erilaisissa valaistusolosuhteissa. Robotti asetettiin aina kohtisuoraan pahvisen levyn eteen n. 110 cm:n päähän siitä, ja IR-kameran näkymä otettiin rvizin avulla näytölle. Kuvista otettiin kuvakaappaukset (kuvat 19, 20, 21 ja 22):



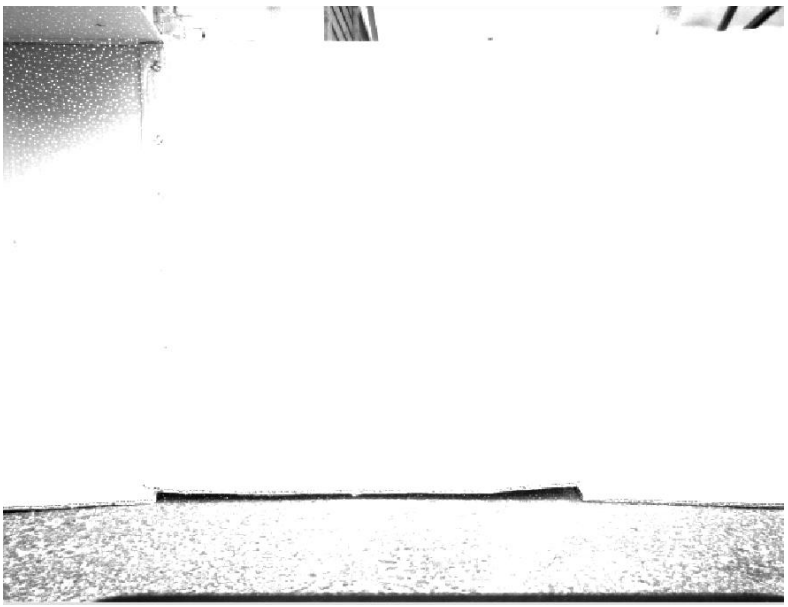
Kuva 19. *Kinect mittauskuvio pimeässä laboratoriossa*

Kuvassa 19 nähdään selvästi mittaukseen käytettävä laserpistekuvio. Kuviossa näkyy Kinectin kameran kautta kuvattuna selvä rakenne, jota ei näkynyt samaa näkymää tavallisella ulkoisella IR-kameralla kuvatessa. Tähän palaamme tämän alaluvun kahdessa viimeisessä kappaleessa. Taustapahvissa erottuu vertikaalisuuntainen taite hieman keskilinjasta vasemmalle. Tumma osa etualalla on lattiaa.



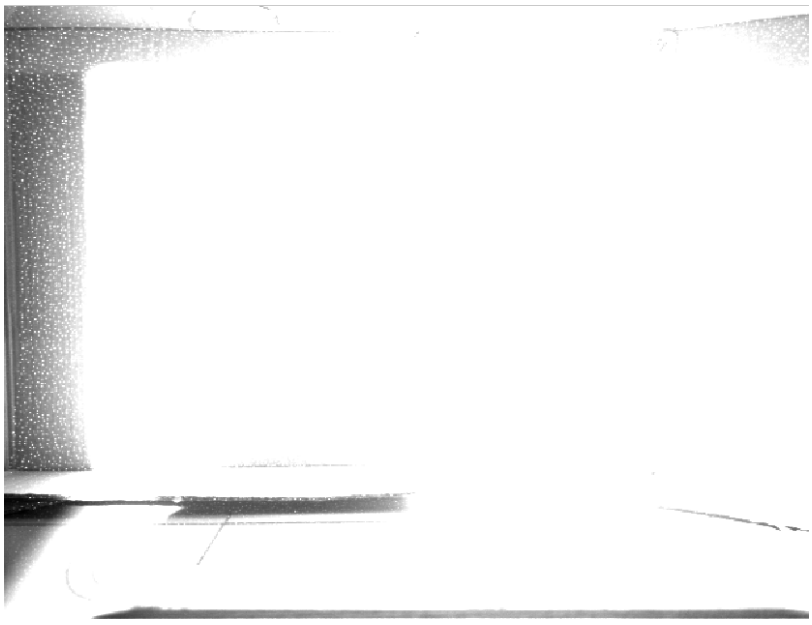
Kuva 20. *Kinect mittauskuvio valaistussa laboratoriossa*

Kuvasta 20 havaitaan, että loisteputkivalossa kuvio pysyy käytännössä saman näköisenä, kuin pimeässä. Laboratoriossa valaistukseen käytettiin Lumilux 3000K Warm White L58W/830 -loisteputkia. Ainakin mainitut loisteputket tuottavat niin vähän infrapunasäteilyä, ettei se häiritse mittausta.



Kuva 21. *Kinect mittauskuvio ulkona, pilvistä*

Kuvan 21 kuvauspäivänä taivas oli paksussa poutapilvessä siten, ettei varjoja voinut erottaa paljain silmin. Kuvasta nähdään, että auringonvalon infrapunasäteily tunkeutuu pilvien läpi ja ”polttaa” kuvan käytännössä pilalle. Pahvilevyn vasemman yläreunan taite on sen verran säteilyn edessä, että sen varjostamalla alueella IR-pisteet näkyvät. Etualalla on karkeaa asfalttia.

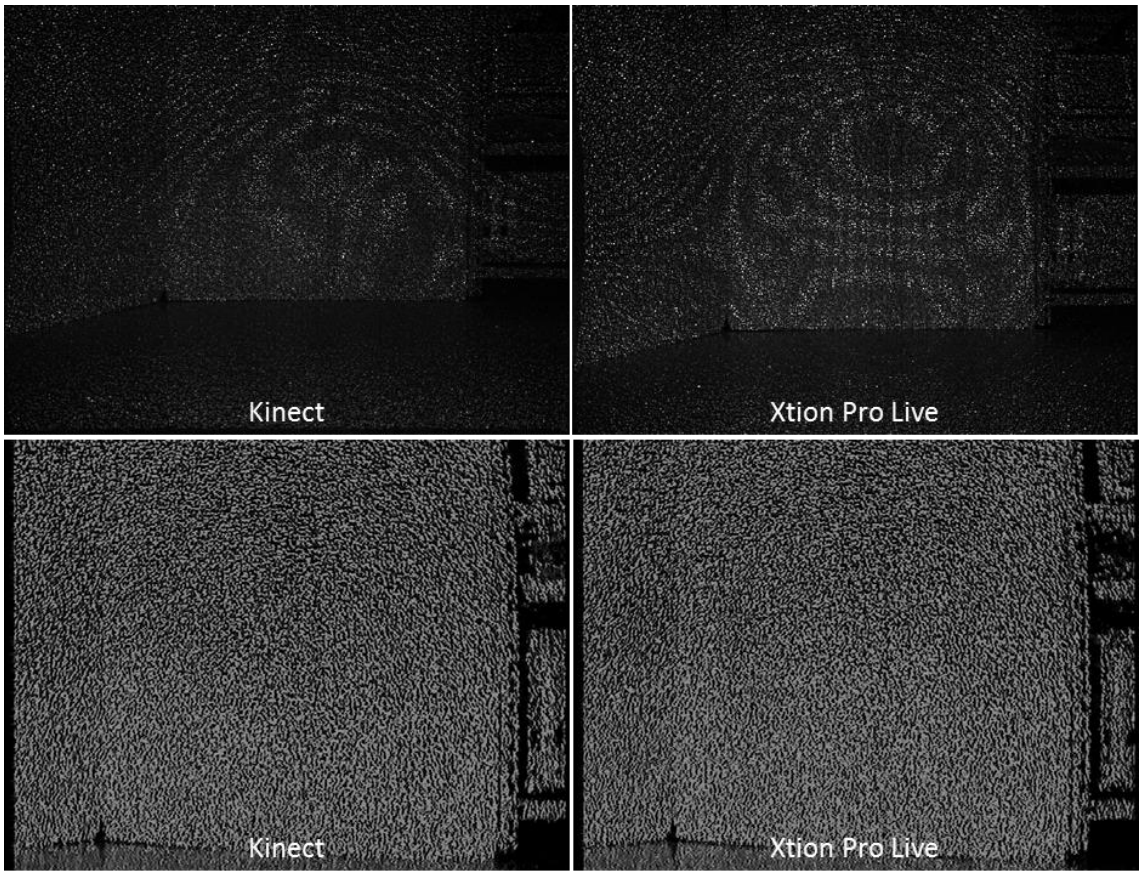


Kuva 22. *Kinect mittauskuvio sisällä, auringonpaiste ikkunasta*

Kuva 22 otettiin sisällä auringon paistaessa ikkunan läpi suoraan pahvilevylle. Kuva on käyttökelvoton koko suoran auringonvalon alueella. Vain ikkunan karmien jättämissä varjokohdissa ja levyn edessä olevien taitteiden alla on nähtävissä pistekuviota.

Edellä olevia kuvia otettaessa meillä oli käytössä vain Kinectillä varustettu robotti. Kinectin ja Xtion Pron laserit toimivat samalla 830 nm:n allonpituudella [104, p. 732], joten auringonvalon vaikutuksen pitäisi olla molemmissa samankaltainen. Tämän varmistamiseksi sensorien kuvia verrattiin puolipilvisellä säällä myötävaloon, jolloin molempien kuvien todettiin olevan koko alueeltaan valkoisia ja siten käyttökelvottomia, myös tähän liitettäviksi.

Lopuksi verrattiin molempien sensorien IR-pistekuvioita rinnakkain. Sitä varten otettiin kuvat taitetusta pahvilevystä molempien sensorien omien IR-kameroiden kautta. Lisäksi kuvattiin molempien pistekuviot ulkoisella IR-videokameralla (JMK-2616 Digital CCD) samassa tilanteessa. Kuvat yhdistettiin yhdeksi, jossa yllä sensorien omalla kameralla ja alla ulkoisella kameralla otetut pistekuviot (kuva 23), etualalla näkyy lattiaa ja oikealla laboratorion hylly.



Kuva 23. *Kinect ja Xtion Pro Live, pistekuviot pimeässä laboratoriossa*

Sensorien omilla kameroilla otetuissa kuvissa erottuu jo aiemmin mainittu rakenne. Ulkoisella kameralla otetuissa sen sijaan ei ole havaittavissa selvää rakennetta. Robottia sivusuuntaan kiertäesssä rakenne muuttaa muotoaan sen mukaan, miten etäisyys mittauspintaan muuttuu. Rakenteen syntymekanismi jäi varmentamatta. Kuten tämän alaluvun aluksi todettiin, ei tarkkaa toimintaperiaatetta ole julkisesti saatavilla. Microsoftin kanssa Kinectin syntyaikoihin yhteistyötä tehneellä PrimeSensellä on kaksi mahdollisesti sen mittausperiaatteeseen liittyvää patenttia, joissa kuvataan useita pistekuvion projisointiin perustuvan etäisyysmittauksen menetelmiä [105], [106]. Rakenteiseen valoon perustuvasta syvyyskuvan muodostuksesta löytyy lisätietoa esimerkiksi lähteistä [104], [107, Ch. 2].

3.7.2 Kartoitus

Turtlebotin kanssa ROSissa käytetään oletuksena kartoitukseen OpenSLAMin (Simultaneous Localization and Mapping) GMapping-algoritmia. Karttaa muodostettaessa GMapping määrittää samalla robotin paikkaa odometrian lisäksi kartoitukseen käytettävien sensorien, kuten laserin, tekemien havaintojen perusteella. [108]

GMapping-algoritmissa käytetään robotin liikemallia, jossa otetaan huomioon odometrian ja ympäristön skannaukseen käytettyjen sensorien virheet. Liikemallin todennäköisyysjakaumasta arvotaan trajektoriehdokkaita. Trajektoriehdokkaista valitaan parhaat

niiden jakautumisen perusteella ja loput hylätään. Hylättyjen tilalle arvotaan painottaen sellaisia, jotka sattuvat jäljelle jääneiden lähistölle. Lopuksi muodostetaan niitä vastaavat karttaehdokkaat ottaen viimeisin mittaustieto huomioon. Samaa jatketaan taas robotin liikuttua pienen matkan eteenpäin, jolloin muodostetaan uusi trajektori- ja karttasukupolvi. Karttaa käytetään tässä satunnaismuuttujana ja algoritmi pyrkii jatkuvasti löytämään todennäköisimmän karttavaihtoehdon. [109], [110], [111], [112]

Kartoittamista varten robottia on ajettava kartoitettavalla alueella niin, että se voi muodostaa koko alueesta kartan sensoritietojensa perusteella. Tämä on helpointa tehdä niin, että sitä ohjataan langattomalla joystickillä, jonka vastaanotin on kytketty robotin päällä olevan koneen USB-porttiin. Tällöin ollaan aina sopivan joystickin kantomatkan päässä, kun robottia ohjataan kävelemällä sen perässä.

Joystickin käyttöä varten on olemassa valmiita merkkikohtaisia käynnistystiedostoja, joista muokkaamalla voi tehdä omalle ohjaimelleen sopivan. Käytössämme oli Logitechin ohjain ja pohjaksi kopioitiin sille tehty yleinen logitech.launch tiedosto, josta muokattiin sopiva. Meidän tapauksessamme piti vain määrittää ”kuolleen miehen kytkin”, jota on painettava jatkuvasti, jotta robotti liikkuu. Kytkimeksi määritettiin ohjaimen ruutu-näppäin lisäämällä parametrilistan jatkoksi rivi [113]:

```
<param name="axis_deadman" value="0" />
```

Kartoitus käynnistetään ajamalla omissa terminaali-ikkunoissaan ensin turtlebot_bringup -paketissa oleva minimal.launch -käynnistystiedosto, sitten jälkeen paketissa turtlebot_navigation sijaitseva gmapping_demo.launch-käynnistystiedosto ja lopuksi käynnistetään joystick-ohjaus ajamalla turtlebot_teleop paketista myjoystick.launch. Suoritettavat komennot ovat:

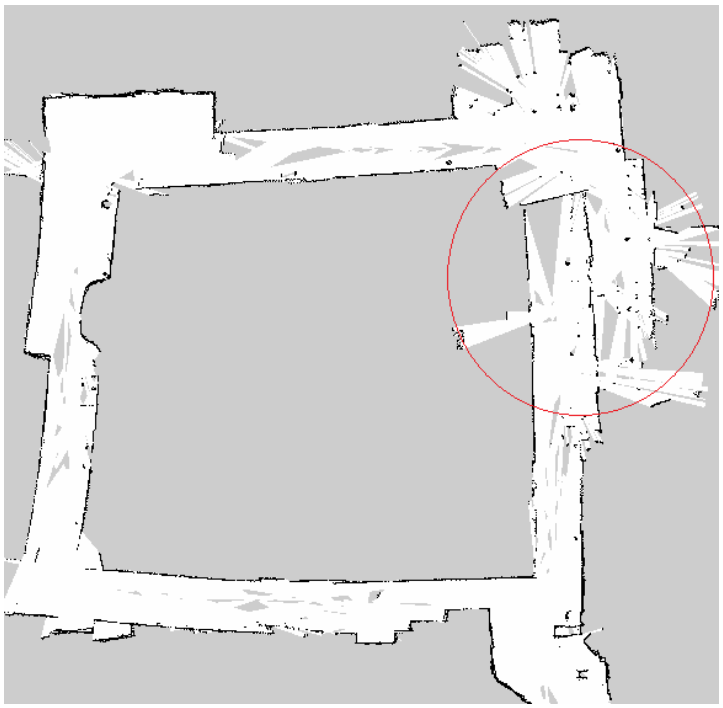
```
roslaunch turtlebot_bringup minimal.launch
roslaunch turtlebot_navigation gmapping_demo.launch
roslaunch turtlebot_teleop myjoystick.launch
```

Käynnistystiedostot kutsuvat sopivin parametrein muita tarvittavia käynnistystiedostoja ja näin saadaan ketjussa käyntiin kaikki tarvittavat nodet oikeilla alkuarvoilla. Kaikki edellä mainitut käynnistystiedostot ajetaan Turtlebot-spesifeistä paketeista ja niissä on sille sopivat perusparametrien arvot.

Hyvänä kartoitus-strategiana pidettiin sellaista, jossa pysähdytään muutaman metrin välein ja pyöräytetään robotti paikallaan ympäri. Jos ympäristössä on paljon yksityiskohtia, pyöräytysten välistä matkaa lyhennetään.

Laboratorio ja sen ulkopuoliset käytävät kartoitettiin Kinectillä varustetulla Turtlebotilla. Useampia karttoja tehdessä huomattiin, että suuri nopeus ei ole karttojen laadun kannalta

hyväksi. Oletusnopeuksilla kartoitettaessa, varsinkin pitkillä käytävillä, pyrkivät käytävien seinät tulemaan kaareviksi ja nurkat todellisesta kulmasta eroaviksi. Tämä johti suljetulla polulla siihen, että käytävän päät eivät aina kohdanneet kartalla, mikä näkyy selkeästi kuvan 24 ympyröidyllä alueella. Kartoissa valkoinen alue on esteistä vapaata tilaa, harmaa tuntematonta aluetta ja musta tarkoittaa estettä.



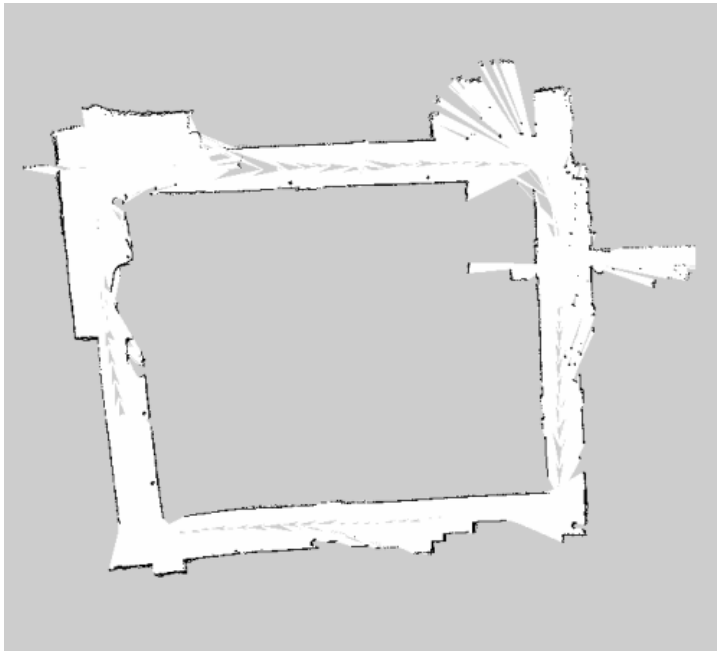
Kuva 24. *Virhe käytävän osien yhdistämisessä, nopeus 1.5 rad/s ja 0.5 m/s*

Nopeuden säätäminen joystickillä oli hankalaa pienen säätöliikkeen vuoksi, joten muutettiin kahta riviä myjoystic.launch-tiedostossa:

```
<param name="scale_angular" value="1.5" />
<param name="scale_linear" value="0.5" />
```

Näistä ensimmäinen säätää robotin kääntymiselle maksiminopeuden radiaaneina sekunnissa ja toinen robotin suoranopeudelle maksimin metreinä sekunnissa. Robotin ohjaamisesta tuli huomattavasti helpompaa, kun sitä voitiin ajaa koko ajan maksiminopeudella.

Kokeilemalla eri arvoja tultiin siihen tulokseen, että optimi kääntymisnopeuden maksimi meidän tapauksessamme on 1.0 rad/s ja vastaavasti suoranopeuden 0.2 m/s. Tällöin kartoitus ei vienyt liikaa aikaa ja toisaalta kartoitusjälki oli riittävän hyvää. Kun samaa aluetta kartoitettiin useaan kertaan näillä asetuksilla, saatiin seinistä yleensä suorempia ja käytävien yhdistäminen onnistui paremmin (kuva 25).

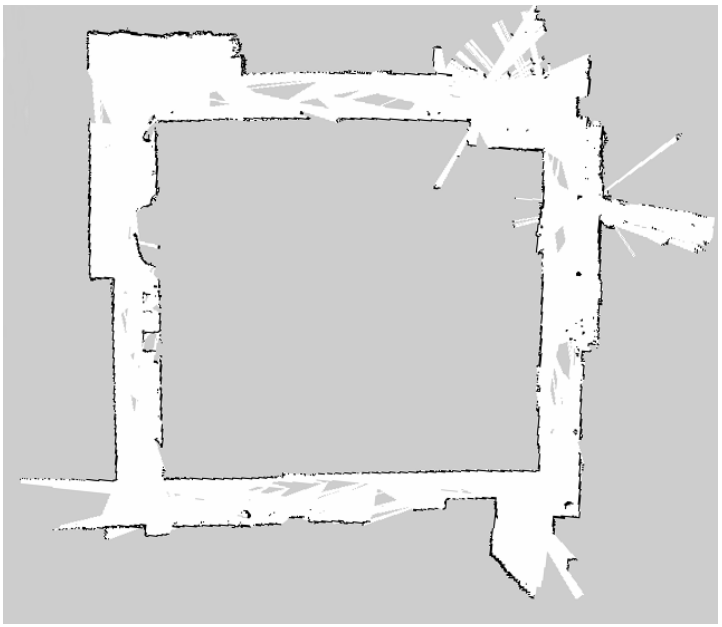


Kuva 25. *Nopeus 1.0 rad/s ja 0.2 m/s*

Kartoituksesta projektin kuluessa kertyneen kokemuksen mukaan pituusvirheiden syntymistä voidaan parhaiten estää ajamalla aina jotakin sellaista pintaa kohti, josta robotti saa jatkuvasti *etenemisen mukaan muuttuvaa* etäisyysdataa. Tämäkään ei tosin aina johtanut toivottuun tulokseen, kun kartoitusalgoritmi korjasi kulmia sivujen ollessa mittavirheiden vuoksi eri pituisia.

Erityisen helposti virhettä aiheuttaa suora käytävä ilman kiintopisteitä, tällöin algoritmi näyttää korjaavan odometrialta saamaansa tietoa skannaustiedon perusteella siten, että käytävistä tulee todellista lyhyempiä. Kun ajetaan suoraan käytävää pitkin, käytävän sivuseinistä mitattava etäisyys ei robotin edetessä muutu, ja tämä näyttää sotkevan tehokkaasti kartoitusta. Suoralla ja tyhjällä käytävällä on hyvä ajaa viistosti aina vuoroin toista seinää kohden, jolloin käytössä on taas koko ajan muuttuva etäisyysdata.

Edelleen kulmat saatiin suoremmiksi, kun kartoittaessa pyöräytettiin robotti ympäri juuri kulman kohdalla. Tällöin se ”näkee” molemmat seinät samaan aikaan kulmasta päin katsottuna ja tämä parantaa yleensä lopputulosta (kuva 26).



Kuva 26. *Kierto kulmissa, nopeus 1.0 rad/s ja 0.2 m/s*

Tässä yhteydessä on mainittava, että kartan onnistumiseen tarvitaan myös onnea. Edellä kerrotuilla tavoilla on helpointa päästä hyvään tulokseen, mutta ne eivät takaa sitä jokaisella kerralla. Kartoituksella on sellainen piirre, että vaikka siinä käytettäisiin samaa dataa, muodostuva kartta ei aina ole samanlainen. Tämä on todettu muun muassa lähteessä [114]. Karttojen muodon vaihtelu on ymmärrettävää, kun otetaan huomioon algoritmin todennäköisyyksiin perustuva toiminta, jota kuvattiin tämän alaluvun alussa. T.Foot (Open Source Robotics Foundation) toteaa saman vastauksessaan lähteessä [114]. Asia on nähtävissä niin ikään Gmappingin ROS-toteutuksessa [115], jossa otantoja varten muodostetaan siemen satunnaisgeneraattoriin tietokoneen kellon perusteella, joka johtaa erilaisiin otantoihin eri ajokerroilla.

Havaittiin myös, että jo kertaalleen karttaan merkityt esteet poistetaan kartoituksen jatkuessa vain, jos niiden takaa saadaan mittauksia toisesta esteestä. Tämä tarkoittaa sitä, että esimerkiksi pitkällä käytävällä vastaan tuleva liikkuva kohde voi aiheuttaa karttaan pitkän esteiden jonon, jollei mittaustuloksia sen jättämän jäljen takaa saada.

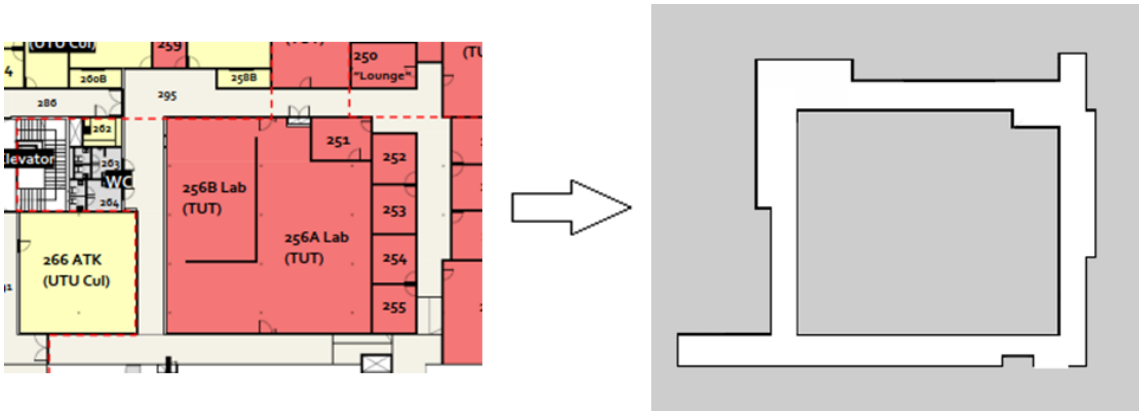
Esteiden merkitsemiselle ja niiden poistamiselle on turtlebot_navigation -paketin gmapping.launch -tiedostossa omat parametrinsa:

```
maxUrange   // (default 6.0 m) sensorin max. käyttökelpoinen kantama
maxRange    // (default 8.0 m) sensorin max. kantama
```

Kartoitettaessa karttaan merkitään kohteet, jotka ovat etäisyyden maxUrange sisällä. Sitä kauempana, mutta maxRangen sisällä olevien mittaustulosten perusteella taas puhdistetaan kartta robotin ja maxUrangen väliseltä alueelta. [108]

Parametrien maxUrange ja maxRange välisen alueen käytölle löytyy luonteva selitys. Ilmeisesti tarkoitus on, että karttaan merkittyjen kohteiden sijaintien tulee olla käyttökelpoisen tarkat ja siksi niiden merkitsemiseen kartalle käytetään vain sensorin tarkempaa mittausaluetta. Kun mittausdataa saadaan tarkan mittausalueen ulkopuolelta, esteen riittävän tarkkaa etäisyyttä ei enää saada, mutta jo mittaustuloksen olemassaolosta voidaan päätellä käyttökelpoisen mittausalueen olevan tuloksen mittaus-suunnassa vapaa esteistä. Näin voidaan myös epätarkempi mittaustieto käyttää kartan parantamiseen.

Kokeilimme vielä kartan tekemistä rakennuksen pohjapiirroksen mukaan. Saimme käyttöömme PDF-kuvat yliopistorakennuksen kaikista kerroksista. Oman kerroksemme kuvasta erotettiin laboratorion ja sitä ympäröivät käytävät sisältänyt osa. Se muokattiin robotin ymmärtämäksi harmaasävykartaksi (kuva 27) ilmaista kuvankäsittelyohjelmaa Gimpiä [116] käyttäen.



Kuva 27. *Kartan muokkaus pohjapiirroksesta*

Kartta on tehtävä mittakaavaan, ja sitä varten tuotiin Gimpiin robotin alueesta aikaisemmin muodostama kartta omalle tasolleen (layer). Tämän jälkeen sovitettiin pohjapiirroksesta tehty kartta samaan mittakaavaan robotin muodostaman kanssa. Aikaansaatu PGM-muotoinen (Portable Graymap) kartta ladattiin robotin koneelle ja sille tehtiin oma YAML-tiedosto edellä skaalaukseen käytetyn kartan YAML:stä. Tällöin ei tarvitse kuin muuttaa YAML-tiedoston nimi sopivaksi ja editoida tiedoston sisältämä kuvatiedoston nimi pohjapiirroksesta tehtyä vastaavaksi. Muokattu YAML-tiedoston rivi:

```
image: /home/user/maps/pohjapiirros.pgm
```

Karttaa kokeiltiin ajamalla robottia sen perusteella ja todettiin mittakaavan olevan kohdallaan. Helppo keino kartan skaalaamiseen on kartoittaa ensin robotilla osa alueesta ja käyttää tätä karttaa pohjapiirroksesta muokatun version skaalaamiseen. YAML-tiedostossa on myös resolution-parametri, joka määrittää kartan pikseliä vastaavan matkan metreinä. Kartta voidaan skaalata siis myös pikselimäärän perusteella, kun esimerkiksi jonkin

pitkän käytävän todellinen pituus tunnetaan. Esimerkiksi GIMPissä voidaan mitata kuvasta etäisyyksiä pikseleinä. Aina ei siis ole välttämätöntä tehdä kartoitusta robotilla ollenkaan.

3.7.3 Navigointi

Robotin käyttämät kartat ovat harmaasävy-bittikarttoja (PGM), kuten jo edellä todettiin. Navigoidessa käytetään kuitenkin useampaa karttatasoa päällekkäin. Silloin staattisen kartoitetun kartan lisäksi ylläpidetään dynaamista global costmapia, johon merkitään myös robotin kulkiessa havaitsemat esteet, joita ei pohjakartassa ole. Reitit lasketaan suuremmassa mittakaavassa global costmapin perusteella, sen lisäksi ajon aikana käytetään local costmapia, jonka avulla lasketaan paikallista polkua robotin lähietäisyydellä dynaamisesti muuttuvan ympäristön mukaan. Käytännössä local costmapin tietojen avulla siis väistetään robotin tielle ilmaantuvat esteet. [117], [118]

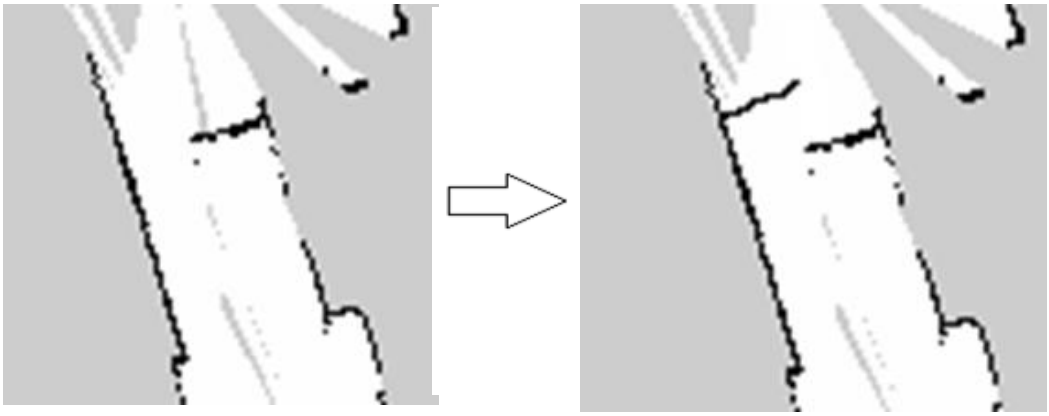
Esteiden merkitsemisetäisyyksille costmapeihin on omat parametrit turtlebot_navigation -paketin costmap_common_params.yaml -tiedostossa:

```
obstacle_range      // (default 2.5) maksimietäisyys jolla este merkitään
raytrace_range      // (default 3.0) etäisyys, johon asti costmap puhdistetaan
```

Parametri obstacle_range kertoo maksimietäisyyden, jolla este merkitään costmapiin. Parametri raytrace_range kertoo etäisyyden, johon asti costmapia puhdistetaan, kun sitä kauempaa saadaan mittaustuloksia.

Kartoitimme aluetta, jossa oli automaattisesti aukeava liukuovi. Sen reagointietäisyys oli toisessa suunnassa suurempi, kuin toisessa. Toisesta suunnasta tullessaan ovi ehti aueta hyvin robotin tullessa, mutta toisessa robotti pysähtyi havaitessaan oven ennen, kuin se ehti aueta. Oven reagointietäisyys oli toisella puolella tarkoituksella määritetty lyhyemmäksi, koska siellä oli vieressä toinen ovi, jonka liikenteen ei haluttu jatkuvasti avaavan liukuovea.

Asia ratkaistiin niin, että oven eteen piirrettiin karttaan virtuaaliseinä, jota kiertäessään robotin oli kuljettava sellaista reittiä, että oven aukaisumekanismi havaitsi sen ajoissa (kuva 28).



Kuva 28. *Kartan muokkaaminen*

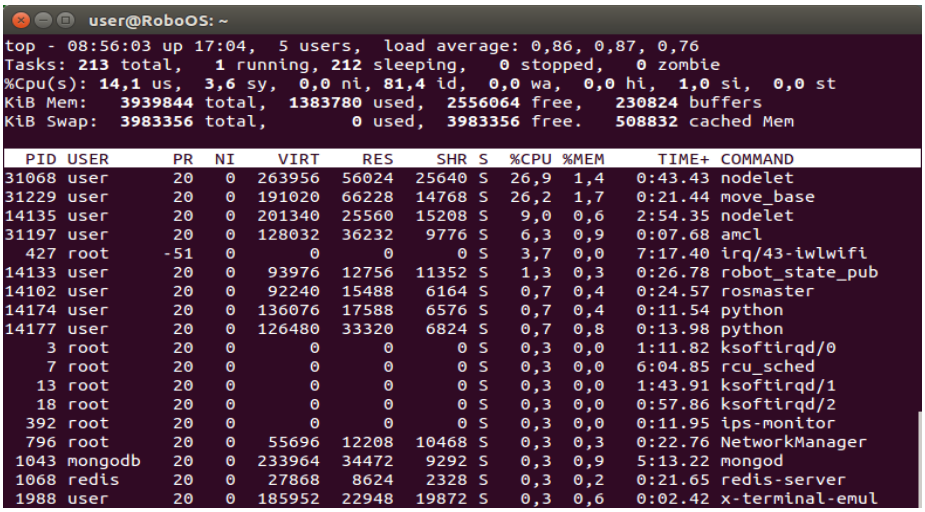
Esteen poistamista sen takaa saadun mittaustuloksen takia ei tarvitse ottaa huomioon virtuaaliseiniä piirtäessä, sillä staattista karttapohjaa ei muuteta niiden perusteella. Tämä todettiin ajamalla virtuaaliseinän vierelle mittaussuunta sen ”läpi” todelliseen seinään. Mittaustulosten saanti todellisesta seinästä todettiin rvizissä local costmap -näkymästä, jossa mittaukset näkyvät selvästi. Tämän jälkeen ei edelleenkään voitu laskea reittiä virtuaaliseinän läpi.

Katsottiin vielä karkeasti kuormitus robotin koneella (Lenovo X201) sekä paikallaan että navigoidessa. Ajettiin turtlebot_bringup -paketin minimal.launch ja turtlebot_navigation -paketin amcl_demo.launch, jotta tarvittavat nodet saadaan käyntiin. Katsottiin ensin prosessorikuorma paikallaan (kuva 29).

```
user@RoboOS: ~
top - 08:58:53 up 17:06,  5 users,  load average: 0,61, 0,86, 0,78
Tasks: 212 total,   2 running, 210 sleeping,   0 stopped,   0 zombie
%Cpu(s):  9,6 us,  4,5 sy,  0,0 ni, 85,6 id,  0,0 wa,  0,0 hi,  0,3 si,  0,0 st
KiB Mem:   3939844 total,  1383820 used,  2556024 free,   230908 buffers
KiB Swap:  3983356 total,        0 used,  3983356 free.   508944 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
31068 user      20   0  263956  56148  25640 S  27,2  1,4   1:30.12 nodelet
31229 user      20   0  191020  66380  14768 S  11,6  1,7   0:53.78 move_base
14135 user      20   0  201340  25560  15208 S   9,0  0,6   3:09.81 nodelet
31197 user      20   0  128032  36232   9776 S   4,3  0,9   0:17.21 amcl
14133 user      20   0   93976  12756  11352 S   1,3  0,3   0:28.99 robot_state_pub
  427 root     -51   0       0      0      0 S   1,0  0,0   7:21.30 irq/43-iwlwifi
    7 root      20   0       0      0      0 S   0,7  0,0   6:05.82 rcu_sched
14102 user      20   0   92240  15488   6164 S   0,7  0,4   0:25.94 rosmaster
14134 user      20   0   94104  13944  10708 S   0,7  0,4   0:04.10 aggregator_node
14150 user      20   0  101112  11152  10168 S   0,7  0,3   0:06.89 nodelet
14156 user      20   0  101112  17352  10240 S   0,7  0,4   0:07.12 nodelet
14177 user      20   0  126480  33320   6824 S   0,7  0,8   0:14.95 python
27801 root      20   0       0      0      0 S   0,7  0,0   4:02.32 kworker/2:1
31222 user      20   0   92916  11428  10444 S   0,7  0,3   0:00.97 nodelet
 1043 mongodb   20   0  233964  34472   9292 S   0,3  0,9   5:14.09 mongod
 1143 root      20   0       0      0      0 S   0,3  0,0   0:00.08 kworker/u8:0
10784 root      20   0       0      0      0 S   0,3  0,0   0:12.20 kworker/3:0
14087 user      20   0   49040  16424   6560 S   0,3  0,4   0:06.15 roslaunch
```

Kuva 29. *Prosessien aiheuttamat kuormat paikallaan*

Kuvasta nähdään, että prosessorien kokonaiskuormitus jää 9.6 %:iin. Muistin kokonaiskulutukseksi saadaan kuvasta laskemalla noin 35 % (Mem: used/Mem: total). Lähes kaikki näkyvät prosessit ovat tunnistettavissa ROS-prosesseiksi. Näitä ovat ainakin nodeletit, move_base, amcl, robot_state_pub, rosmaster, aggregator_node ja roslaunch. Seuraavaksi katsottiin tilanne robotin navigoidessa kohteeseen (kuva 30).



Kuva 30. *Prosessien aiheuttamat kuormat navigoidessa*

Tästä nähdään, että prosessorien kokonaiskuormitus on kasvanut maltillisesti 14.1 %:iin ja muistinkulutus edellisen tapauksen tapaan laskien on edelleen luokkaa 35 %. Molemmissa tapauksissa lukemat luonnollisesti vaihtelivat hieman, mutta kuvakaappausten tilanne oli tyypillinen. Navigointia ajatellen roboteissa käytettävien koneiden prosessori- ja muistikapasiteetti vaikuttaa riittävältä.

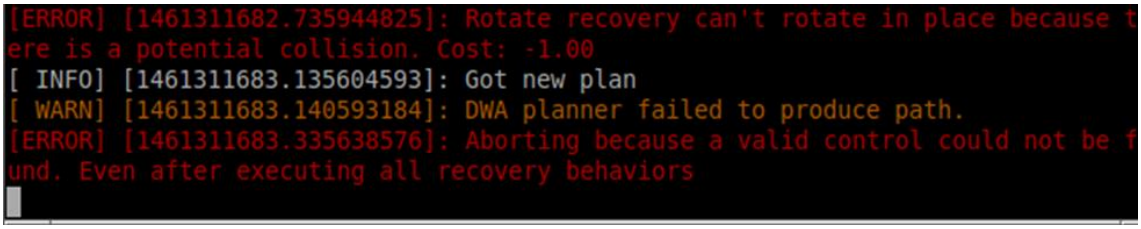
Robotin kykyä löytää vaihtoehtoisia polkuja annettuun pisteeseen kokeiltiin ajamalla sitä laboratoriossa siten, että suljettiin sen global costmapin perusteella laskema polku. Tällöin voitiin todeta, että jos vaihtoehtoinen polku costmapin mukaan oli olemassa, se löydettiin ja robotti ajoi kohteeseen toista reittiä.

Robotin kulkureittejä tukittiin myös sijoittamalla esteitä sen laskemille vaihtoehtoisille reiteille, ja poistamalla jokin este robotin jo kokeilemilta reiteiltä. Ongelmia ilmaantui, jos näitä vapautuneita reittejä ei ollut poistettu global cosmapista esimerkiksi niiden ohi ajaessa. Vaikka vapaita reittejä olisi tällöin todellisuudessa ollut käytettävissä, ei robotti välttämättä kyennyt enää laskemaan reittiä kohteeseen lainkaan global costmapin väärän informaation vuoksi. Kuten jo aiemmin on todettu, esteet poistuvat costmapeista, mikäli niiden takaa saadaan mittaustuloksia.

Robotin kykyä väistää dynaamisia esteitä kokeiltiin asettamalla niitä robotin kulkureitille. Tilannetta kiristettiin asettamalla este aina vähän lähemmäksi kulkevaa robottia. Mikäli tilaa oli riittävästi ja este havaittiin kaukaa, noin 2 m:n etäisyydelta, osasi robotti yleensä väistää sen sujuvasti. Mikäli este laitettiin sen eteen lyhyemmän matkan päähän, mutta kuitenkin minimittausetäisyyden ulkopuolelle (3.7.1), onnistui väistäminen yleensä kohtalaisesti vaatien joskus useammankin ympäriskannauksen.

Kokeiltiin vielä dynaamisen esteen joutumista robotin minimihavaintoetäisyyden sisäpuolelle. Käytännössä tällaiseen tilanteeseen voitaisiin joutua esimerkiksi silloin, kun ihminen seisoo aivan robotin takana sen kääntyessä ympäri ja lähtiessä ajamaan.

Robotin kulkiessa suoraan sen eteen vietiin nopeasti este. Robotti törmäsi suoraan siihen ja pysähtyi puskurikytkimen painuessa alas. Sitten se yritti uudelleen liikkeelle ja jäi pitkäksi ajaksi toistamaan tätä paikalleen. Hieman vaihdellen noin minuutin yrittämisen jälkeen se lopulta jumiutui paikalleen, eikä toipunut siitä enää omin voimin. Syynä oli virhe paikannuksessa, jolloin robotti luuli olevansa esteen päällä, eikä siksi yrittänyt toipua tilanteesta skannaamalla ympäristöä (kuva 31).



Kuva 31. *Virheilmoituksia robotin jumiutuessa*

Jos esteen poisti ennen jumiutumista, robotti jatkoi matkaa. Jos robotti siis törmää esimerkiksi ihmiseen, joka väistää sitä heti sen jälkeen, robotti toipuu tilanteesta ja jatkaa matkaa.

Kun robotti jumiutui paikalleen niin, ettei se voinut tehdä ympäriskannauksia, sitä ei voinut ajaa mihinkään suuntaan ilman, että sen kävi nostamassa hieman sivuun ja osoitti sille uuden alkuposition. Testikäyttöliittymällä ei voi antaa alkupositiota, vaan siihen on käytettävä rviziä. Toinen vaihtoehto on sammuttaa navigointi ja viedä robotti alkupisteeseen ja käynnistää siellä navigointi uudelleen.

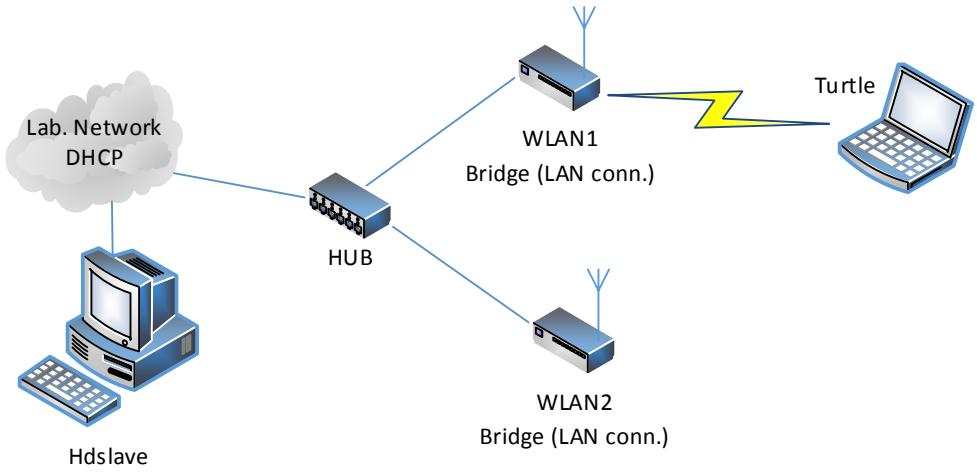
Kokeiltiin myös robotin liikkumisen estämistä ahtaalla, noin 140 cm levyisellä, käytävällä astumalla ensin sen eteen ja sitten aina samaan suuntaan, mihin se väisti. Tällöin robotti jumiutui yleensä 2-3 väistöliikkeen jälkeen samalla tavoin, kuin edellisessä tapauksessa. Tässäkin tapauksessa robotti toipui tilanteesta, jos ajoissa jäi paikalleen ja antoi sen jatkaa matkaansa.

3.7.4 Tiedonsiirto

Robotin käyttäessä langatonta tiedonsiirtoa osattiin jo etukäteen odottaa ongelmia, varsinkin laajoilla toiminta-alueilla. Tämä konkretisoitui erityisesti tukiasemapeittoa toisella tukiasemalla jatkettaessa. Tukiaseman vaihtamiseen kuluva aika muodostui tässä olennaiseksi tekijäksi, sillä IP-osoitteen menettäminen ROS-ympäristössä merkitsee TCP/IP -sanomien kulun estymistä myös robotin koneen sisäisesti.

Halusimme tietää, kuinka paljon tukiaseman vaihtamiseen kului aikaa. Mittaukset katsottiin parhaaksi hajottaa niin, että kaikkea ei tarvitse tehdä samaan aikaan. Tukiasemina käytettiin TP-Link Archer C2600:ia ja roamaavana koneena kannettavaa Lenovo X201:tä (Turtle). Varsinkin robotin siirtely WDS-mittauksessa kärryillä edestakaisin läpi ovien samalla mitaten olisi ollut kovin hankalaa. Tästä syystä tukiaseman vaihtoajat mitattiin

pelkän kannettavan tietokoneen avulla. Mittaus suoritettiin kirjaamalla tukiaseman vaihtamisen aikana kadonneiden pingien (väli 1 s) määrä molemmissa luvussa 3.5 esitetyissä kytkennöissä. Ensin mittaus tehtiin tukiasemat siltana (kuva 32).



Kuva 32. *Roaming-mittaus, tukiasemat siltana*

Tukiasemien välillä on tässä lankayhteys, eikä niiden kesken saatu riittävää etäisyyttä, jotta tukiaseman vaihtamista olisi voitu kokeilla liikuttamalla Turtlea niiden välillä. Mittaus tehtiin siten, että ensin laitettiin Turtle lähettämään pingiä Hdslavelle ja sitten sammutettiin tukiasema, jossa se oli kirjautuneena. Kun Turtle oli siirtynyt toiseen tukiasemaan ja sanomat kulkivat taas, sammutettiin ping ja kirjattiin kadonneiden pakettien määrä. Sitten kytkettiin tukiasemaan virta takaisin ja tehtiin sama toiseen suuntaan. Tulokset esitetään taulukossa 12.

Taulukko 12. *Kadonneet paketit, tukiasemat siltana*

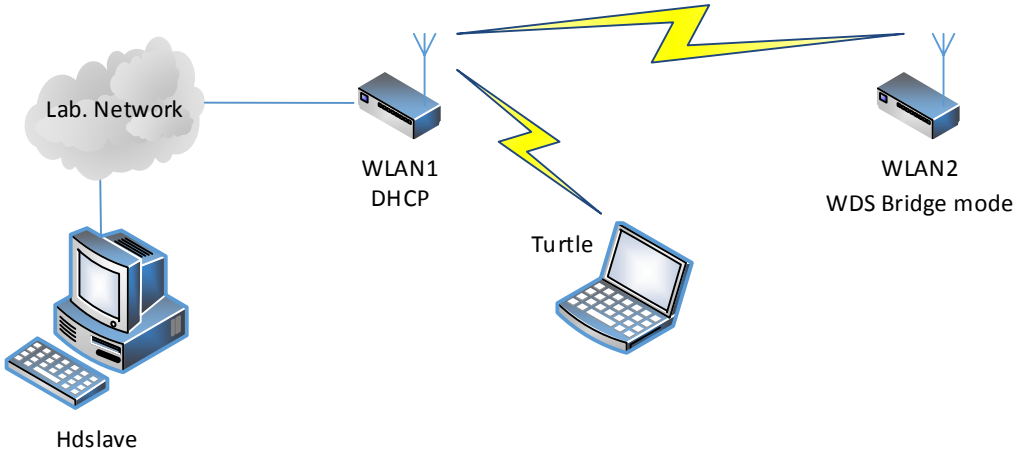
Mitt. n:o	**Virta pois/ packet loss**
1	5
2	1
3	16
4	1
5	5
6	2
7	4
8	2
9	15
10	4

Kuten taulukosta nähdään, tukiaseman vaihdon yhteydessä kadonneiden pakettien määrä vaihteli suuresti. Koska ping-pakettien lähetyksen väli oli 1 s, kadonneiden pakettien lukumäärä on sama, kuin vaihdon viive yhteydestä yhteyteen sekunteina ±1 s.

Sanomalokia tarkastellessa nähtiin, että viive yhteyden katkaisemisesta ensimmäiseen Probe Request -sanomaan vaihteli aina yli kahteen sekuntiin asti. Kun Probe Response saatiin toiselta tukiasemalta, autentikointia ei välttämättä yritetty ensimmäisellä kerralla

ja silloin kun yritettiin, viive sen aloittamiseen viimeisestä Probe Responsesta saattoi olla luokkaa kaksi sekuntia. Pahimmillaan autentikointi saattoi ensimmäisellä kerralla epäonnistua. Avaintenvaihdon aikana joku sanoma saattoi puuttua välistä ja joitakin sanomia uudelleenlähetettiin. Kun kaikki tämä sattui sopivasti kohdalleen, viive saattoi kasvaa varsin suureksi, kuten taulukon 12 arvoista 3. ja 9. mittauksessa nähdään.

Seuraavaksi mitattiin menetettyjen pakettien lukumäärä, kun tukiaseman peittoa jatkettiin langattomasti WDS-siltana. Kytkentä esitetään kuvassa 33.



Kuva 33. *Roaming-mittaus, tukiasemapeitto jatkettu WDS-sillalla*

Tässä saatoimme sijoittaa WLAN-peittoa jatkavan tukiaseman vapaasti ja näin ollen mittaukset tehtiin liikuttamalla Turtle-konetta tukiasemien välillä. Asemaa, jossa kone oli kiinni, monitoroitiin komennolla:

```
watch -n 1 ”iwconfig | grep Access”
```

Komennolla tulostetaan 1 s välein iwconfigin tulosteesta riviä, jolla tukiaseman (Access Point) MAC-osoite näkyy. MAC-osoitteen loppuosaa seuraamalla näki, missä tukiasemassa kone kulloinkin oli kiinni. Konetta kuljetettiin tukiasemien välillä ja tukiaseman vaihdon yhteydessä kadonneet paketit kirjattiin (taulukko 13) samaan tapaan, kuin edellisessä mittauksessa.

Taulukko 13. *Kadonneet paketit, tukiasemapeitto jatkettu WDS-siltana*

Mitt. n:o	**tukiaseman vaihto/packet loss**
1	8
2	0
3	2
4	7
5	10
6	5
7	12
8	5

9	1
10	15

Taulukosta nähdään, että tulokset olivat samansuuntaisia kuin edellisessä mittauksessa. Pakettien hävikin vaihtelu oli tässäkin huomattavan suuri.

Halusimme saada käsityksen siitä, kuinka pitkiä yhteyskatkoja navigointiin tarvittavat ROS-prosessit sietävät ennen kuin niitä alkaa sammua. Tämän selvittämiseksi otettiin aikaa sekuntikellolla verkkoyhteyden katkeamisesta siihen, kun ensimmäiset ROS-prosessit putosivat pois. Tässä Turtle-kone (Lenovo X201) yhdistettiin robottiin, jotta kaikki ROS-prosessit saatiin käyntiin.

Tarvittavat nodet käynnistettiin ajamalla omissa terminaaleissaan sekä minimal.launch että amcl_demo.launch ja katkaisemalla tukiasemasta virta vaihtelevan ajan kuluttua siitä, kun käynnistykset oli suoritettu loppuun. Mitattiin aika virran katkaisusta siihen, kun ensimmäiset prosessien putoamisesta kertovat ilmoitukset saatiin terminaaleihin. Mitatut ajat on esitetty taulukossa 14.

Taulukko 14. *ROS-prosessien pysyminen käynnissä verkkokatkon aikana*

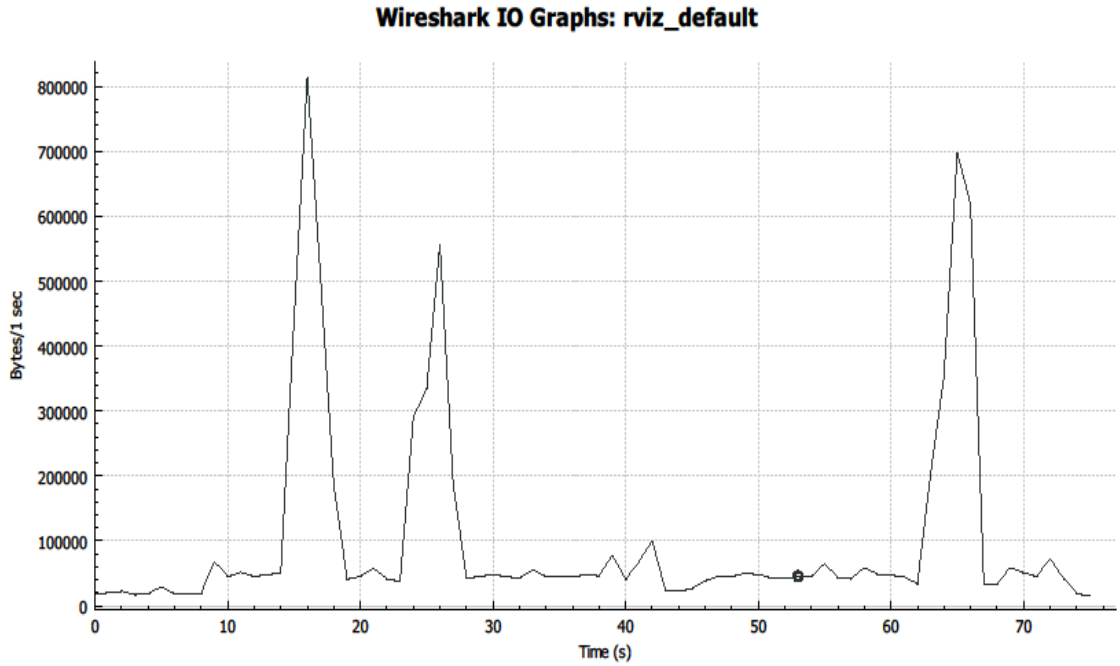
Mitt. n:o	**Aika [s]**
1	7.532
2	8.289
3	6.616
4	8.105
5	7.860
6	5.821
7	8.557
8	7.655
9	9.564
10	5.875
	ka. ≈ 7.6

Käsiajanotolla mitatut ajat ovat suuntaa-antavia, mutta niistä selviää käytettävissä oleva likimääräinen aika. Voidaan todeta, että tukiasemalta toiseen siirtymisen pitäisi onnistua vielä 5 s:n vaihtoajalla, vaikka mittaajan reaktioaika otettaisiin tuloksissa huomioon.

Verrattiin vielä siirtotien kuormia kahdessa erilaisessa tapauksessa. Ensin robottia ohjattiin rvizillä kaikki oletusvalinnat käytössä, jolloin siirtotiellä kulkee muun muassa amcl-partikkelit, robotin sijainti, laserskannauskuvio, pohjakartta sekä global- ja local costmap. Liikenne mitattiin Wiresharkilla [119] ottamalla WLAN-tikulla kaikki käytetyn kanavan liikenne talteen ja suodattamalla siitä pelkkä robotin koneelle ja sieltä ulos menevä liikenne. Suodattimena käytimme Turtlen WLAN-kortin MAC-osoitetta ja suodatin oli muotoa wlan.addr==58:94:6b:a9:79:e0.

Mittaukset tehtiin ajamalla kahden karttapisteen väliä kymmenen kertaa edestakaisin ja ottamalla liikenne talteen jokaiselta edestakaiselta matkalta. Tarkoitus oli laskea näistä

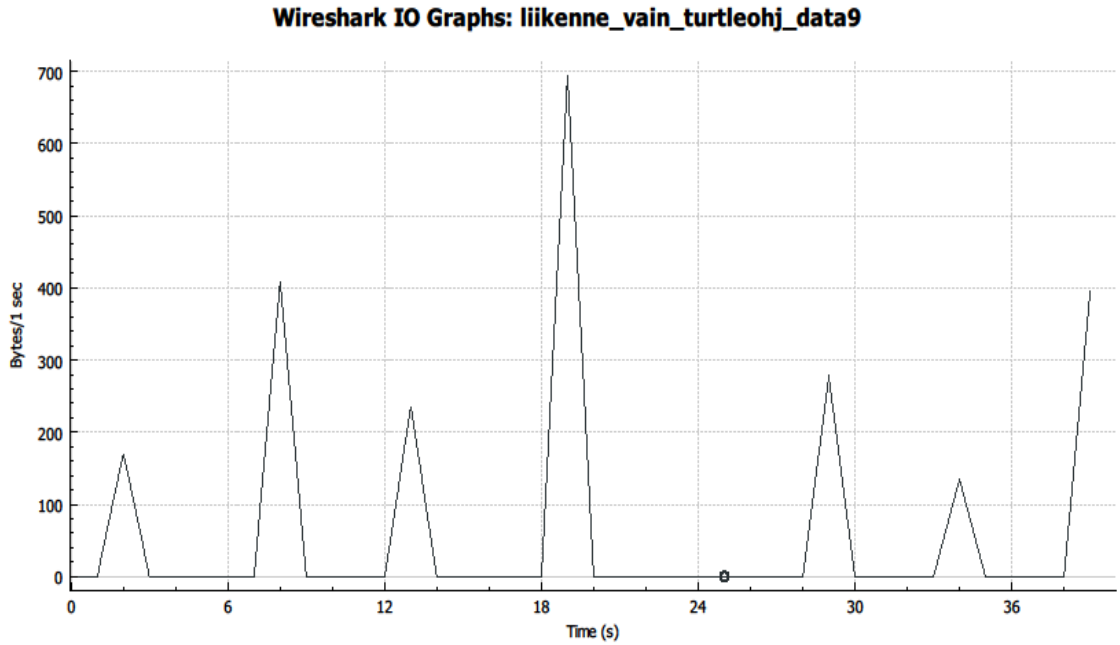
mittauksista keskiarvokäyrä. Kun eri mittauskertojen liikennekuvaajia verrattiin keskenään, havaittiin, että kuorma vaihteli suuresti. Kuvaajissa olevat liikennepiikit sijoittuivat eri ajokerroilla täysin eri paikkoihin ja niiden määrä vaihteli. Tästä syystä keskiarvokuvaajan laskemisesta luovuttiin ja tässä esitetään mitatuista yksi tyypillinen kuvaaja, josta saa käsityksen liikennemääristä rvizin avulla oletusasetuksin navigoidessa (kuva 34).



Kuva 34. WLAN liikenne, rviz

Maksimikuormat olivat suhteellisen suuria, mikä on ymmärrettävää, kun etäkoneelle on siirrettävä paljon dataa näytölle esitettäväksi. Eri ajokertojen suurimpien piikkien arvot vaihtelivat 339.4 ja 869.4 kB/s välillä keskiarvon ollessa 680.0 kB/s. Kuvassa näkyvä tasaisempi kuorma piikkien välillä oli kaikissa mittauksissa 50 kB/s tuntumassa.

Toisessa mittauksessa käytettiin tässä työssä toteutettua testikäyttöliittymää. Mittaus tehtiin edellisen tapaan, mutta ajamalla tällä kertaa viisi kertaa kahden pisteen väliä edestakaisin ja ottamalla liikenne talteen erikseen molemmilta ajosuunnilta. Jälleen kuormien liikennepiikit samansuuntaisilla ajokerroilla osuivat täysin eri kohtiin ja vaihtelivat muutenkin samaan tapaan, kuin edellisessä mittauksessa. Myös tässä esitetään mitatuista vain tyypillinen kuvaaja (Kuva 35).



Kuva 35. *WLAN-liikenne, testikäyttöliittymä*

Tässä mittauksessa nähtiin odotetusti huomattavasti pienempiä kuormia, koska testikäyttöliittymä ei tilaa mitään topiceja. Se julkaisee vain kahteen topiciin, toiseen määränpään koordinaattisanomia ja toiseen pysäytyssanomia. Tässä mittauskerran suurimman piikin arvo vaihteli 0.374 ja 7.343 kB/s välillä keskiarvon ollessa 2.384 kB/s. Tasaisempi kuorma piikkien välillä oli kaikissa mittauksissa 0 kB/s. Vaikka liikennettä oli näin vähän, ylitti sen määrä kuitenkin odotukset. Liikennettä oli paljon siihen nähden, että käyttöliittymä julkaisee vain yhden sanoman koko mittauksen aikana. Lähes kaikki liikenne on siis ”tyhjäkäyntikuormaa”, kuten yhteyden kontrollisanomia.

4. PÄÄTELMÄT

Tässä luvussa käsitellään sensorien mittaustulokset. Tuloksia verrataan keskenään, sekä niiden luvussa 2.7 esitettyihin teknisiin ominaisuuksiin. Lisäksi kootaan yhteenveto kartoituksesta saaduista kokemuksista ja esitetään sen onnistumisen kannalta tärkeitä seikkoja. Robotin navigointikykyä arvioidaan suoritettujen testien perusteella ja lopuksi tehdään lyhyt yhteenveto tiedonsiirron mittauksista ja arvioidaan käytetyn WLAN-verkon soveltuvuutta robottiympäristöön.

4.1 3D-sensorit

Robotin toiminnan kannalta sensorin mittausalueen tulisi olla mahdollisimman laaja ja kattaa myös aivan lähietäisyydet. Ympäriskannaavalla laserilla on molemmat ominaisuudet, mutta silloin hinta muodostuu hankintaa rajoittavaksi tekijäksi.

Minimimittausetäisyyksissä sensorit olivat suhteellisen tasaveroisia, Kinectin ollessa hieman Xtion Prota parempi. Sensorit on kiinnitetty robotteihin hieman keskilinjan taakse. Kinect 8.3 cm ja Xtion Pro 8.7 cm keskilinjasta sensorin etupintaan. Turtlebotin halkaisijan ollessa 35.0 cm, mittaustuloksista voidaan laskea sensorien lähietäisyyden katvealue mitattaessa näihin materiaaleihin. Katvealueen pituus pyöristetään mittaustulosten tarkkuuteen (± 0.25 cm). Kinectillä katvealue vaihteli siis välillä 24.2 – 39.2 ≈ 24.0 – 39.0 cm ja Xtion Prolla välillä 22.8 – 51.3 ≈ 23.0 – 51.5 cm.

Katve robotin lähietäisyydellä heikentää oleellisesti robotin kykyä toimia ahtaassa dynaamisessa ympäristössä. Robotin pitäisi voida tehdä havaintoja heti omien dimensioidensa ulkopuolelta, jotta kaikki lähiesteet voitaisiin väistää. Mitattuja minimimittausetäisyyksiä ei näin ollen voida pitää riittävinä.

Maksimimittausetäisyyksissä sensorien keskinäinen ero oli suurempi. Maksimietäisyydet koko matkaltaan staattiselle laserviivalle vaihtelivat Kinectillä välillä 5.7 – 6.1 m ja Xtion Prolla välillä 5.8 – 7.1 m. Aiemmin käsiteltyjen (3.7.2) maxUrange- ja maxRange-parametrien oletusarvot sopivat molemmille sensoreille lasikuitutapettia haastavammaksi mitattavaksi osoittautuneeseen valkoiseen oveen mitatessa saatuihin tuloksiin.

Näistä maxUrange (default 6.0 m), joka tarkoittaa karttaan merkittävän kohteen maksimietäisyyttä, on lähellä molempien sensorien oveen mitattuja tuloksia. Sama on nähtävissä toisen parametrin, maxRangen (default 8.0 m) tapauksessa. Oveen mitatessa Kinectille saatiin 7.9 ja Xtion Prolle 7.8 metrin vielä staattisia osuuksia sisältävän laserviivan maksimietäisyydet. Parametri ilmaisee maksimietäisyyden, josta saatavilla mittaustuloksilla puhdistetaan robotin ja maxUrangen välillä oleva kartan alue. Parametrien oletusarvot Turtlebotille on mittausten mukaan määritetty varsin sopiviksi.

Kinect selviytyi kuitenkin hieman huonommin juuri maxUrangen ja maxRangen välisellä alueella, jossa sillä oli 7 metrin etäisyyden ympäristössä alue, jolla ei saatu staattista mittaustietoa ollenkaan. Myös käytännön kartoituksessa Xtion Pro todettiin näistä kahdesta sensorista paremmaksi. Kinectillä tehdyissä kartoissa oli kartoitetuilla alueilla enemmän hajanaisia harmaita säikeitä, jota epätarkan vyöhykkeen osuminen 6 ja 8 metrin välille osaltaan selittää.

Valaistusolosuhteet vaikuttavat kokeiden perusteella oleellisesti sensorien käyttökelpoisuuteen eri ympäristöissä. Sisäkäytössä tavalliset loisteputket eivät normaalisti haittaa mittausta, mutta mikäli auringonvalon sisältämä infrapunasäteily pääsee suoraan mitattaville pinnoille esimerkiksi ikkunan läpi, se voi sokaista sensorin. Ulkokäytössä auringonvalon edessä on oltava jokin este, muutoin navigointi ei onnistu. Kuten kokeessakin todettiin, pilvet auringon edessä eivät välttämättä siihen riitä.

4.2 Kartoitus

Hyvän kartan aikaansaaminen robottia ajamalla ei aina ole helppoa kummallakaan sensoreistamme. Sensorien lyhyt maksimihavaintoetäisyys ja rajattu mittauskeila johtavat siihen, että robottia on vähän väliä pyöräyttävä ympäri koko ympäristön mittaamiseksi. Pienissä tiloissa ongelmaa ei yleensä ole, mutta suuremmat tilat ja erityisesti pitkät suljetut polut aiheuttavat vaikeuksia.

Turtlebotilla, joka on varustettu Kinectin ja Xtion Pron kaltaisella sensorilla, kartoitettaessa on syytä muistaa seuraavat seikat:

- Kartoitusnopeuden tulee olla maltillinen, meille sopivin oli kääntymisnopeus 1.0 rad/s ja suoranopeus 0.2 m/s.
- Robotin tulee eteenpäin liikkuessaan saada koko ajan etäisyyden mukaan muuttuvaa mittausdataa. Tämä on suoralla ja tyhjällä käytävällä saavutettavissa ajamalla viistosti vuoroin kumpaakin käytävän seinää lähestyen.
- Kulmissa robotti on hyvä pysäyttää noin metrin päähän kulmasta ja pyöräyttää se ympäri niin, että saadaan mittaustietoa molemmista seinistä samaan aikaan.

Kartan tekeminen pohjapiiroksen avulla on myös hyvä vaihtoehto, erityisesti laajoilla toiminta-alueilla. Mahdolliset pohjapiirroksen muutokset, kuten uudet seinät tai seinän poisto on helposti muokattavissa karttaan. Myös silloin, kun ympäristössä on liikuteltavia esteitä, kuten esimerkiksi potilassänkyjä ja tuoleja sairaalan käytävällä, on hyvä tehdä kartta pohjapiirroksesta. Tällöin robotti kohtelee kaikkia löytämiään esteitä dynaamisina ja merkitsee ja poistaa niitä costmapeista tilanteen muuttuessa.

4.3 Navigointi

Robotin navigointi sujuu varsin hyvin, jos tilaa on paljon ympärillä. Robotti osasi laskea vaihtoehtoisen reitin, jos ensin laskettu oli tukossa. Myös sen eteen tulleiden dynaamisten esteiden väistäminen sujui niin kauan hyvin, kun se pystyi mittaamaan ne. Mikäli sen minimimittausetäisyyden sisäpuolelle joutui esteitä, robotti oli useimmiten vaikeuksissa.

Haasteena olivat myös jo alueelta poistuneiden dynaamisten esteiden jäljet aivan robotin lähituntumassa. Koska robotti määrittää sijaintiaan jatkuvasti ympäristöä mitatessaan, myös sen laskema todennäköinen sijainti heittelee hieman. Kun laskettu sijainti sattuu tällaisen jäljen päälle, robotti ei enää toivu tilanteesta, vaan siihen on operaattorin fyysisesti puututtava.

Dynaamisten esteiden poistamiseen costmapeilta vaikuttaa suuresti sensorin kantomatka. Mikäli sensorin kantomatka ei riitä havaintojen saamiseen esteen jättämän jäljen takaa, ei sitä osata poistaa.

Navigointiin voidaan vaikuttaa muokkaamalla pohjakarttoja. Robotti ei laske reittiä sellaiselle alueelle, joka on erotettu virtuaaliseinällä muusta alueesta. Tästä on hyötyä, mikäli robotin toiminta-aluetta halutaan rajata. Robotin laskemaan reittiin voidaan myös vaikuttaa virtuaaliseinillä ja näin ohjata se kulkemaan haluttua polkua. Tämän tuoma hyöty osoitettiin, kun robotti ohjattiin virtuaaliseinän avulla kulkemaan automaattioven reagointietäisyyden sisäpuolelle sen avaamiseksi.

Kartoituksen tapaan myös navigoinnissa suurimmaksi rajoittavaksi tekijäksi nousee siis sensorien suorituskyky. Tässä kokoonpanossa robottia voidaan ajaa turvallisesti ympäristöissä, joissa se ei joudu dynaamisesti ahtaisiin paikkoihin. Käytännössä tämä tarkoittaa sitä, että esimerkiksi ruuhkaisilla käytävillä navigointi on epävarmamaa.

4.4 Tiedonsiirto

Tiedonsiirto toisella tukiasemalla laajennetulla WLAN-verkolla toimi huonosti. Kun otetaan luvussa 3.7.4 tehtyjen mittausten tarkkuus huomioon ja oletetaan suurimmaksi sallituksi roaming-ajaksi 5 s, nähdään, että tukiasemavaihto olisi onnistunut 6-8 tapauksessa 10:stä kun molemmat tukiasemat toimivat siltana. WDS:n tapauksessa vain 3-5 tapauksessa.

Käytännössä tällä laitteistokokoonpanolla tässä ympäristössä on vaikea ajaa robottia usean WLAN-tukiaseman alueella, koska on lähes väistämätöntä, että jossakin vaiheessa roamingiin kuluva aika ylittää ROS-prosessien sietokynnyksen. Tämä tuli robottia operoidessa todetuksi myös käytännössä useaan kertaan sekä omilla (TP-Link Archer C2600) että laitoksen TUT-verkon tukiasemilla (Cisco AIR-AP 1252AG-E-K9).

Mittausympäristö oli varsin ruuhkainen, matkapuhelimen (samsung S4) WiFi-analysaattorilla (farproc Wifi Analyzer) nähtiin tukiasemia olevan sen havaintoalueella yli 20 keskitettyinä kanaville 1, 6 ja 11 overlappingin estämiseksi. Rauhallisimmalla kanavalla 6, jolla mittaukset tehtiin, oli samaan aikaan 5-6 muuta tukiasemaa. Ruuhkainen ympäristö saattoi vaikuttaa mittauksiin niiden tulosta heikentävästi. Robotin kykyä vaihtaa tukiasemaa on syytä testata kohdeympäristössä kulloisillakin robotissa käytettävillä tietokoneilla.

Parempi tapa verkon laajentamiseksi voi olla esimerkiksi kiinnittää robotti tiettyyn tukiasemaan ja varustaa se WLAN-tikulla, jossa on hyvä ulkoinen antenni. Toinen mahdollisuus on tutkia, millaista hyötyä roaming aggressiveness-parametrista on käytännössä. Myös 3G/4G-mobiiliverkkoa voisi harkita, mikäli sillä on alueella hyvä peitto.

Tällä hetkellä ainoa varma keino välttää OpenCRP:ssä WLAN-tukiaseman vaihdosta johtuvat katkokset laajalla toiminta-alueella on ajaa robottia ilman verkkoa. Tämä onnistuu asettamalla ROSin IP-osoitteet .bashrc:ssä localhosteiksi:

```
export ROS_MASTER_URI=http://localhost:11311
export ROS_HOSTNAME=localhost;
```

Jos robottia ajetaan ilman verkkoa, joudutaan käyttöliittymää käyttämään paikallisesti robotin päällä olevassa koneessa. Tällainen mahdollisuus on hyvä olla olemassa, mutta etäohjausmahdollisuuden menettäminen hankaloittaa käyttöä. Siksi olisi suotavaa sovittaa käytettävä tiedonsiirtomenetelmä robotin toimintaympäristön mukaan.

5. YHTEENVETO

Tämän työn päämääränä oli OpenCRP -demonstraatio- ja pilotointiympäristön rakentaminen yhdelle mobiilirobotille. Tarkoitus oli myös kerätä tietoa, paitsi ympäristön toteuttamiseen, myös sen käyttöön liittyvistä haasteista.

Ympäristö rakennettiin, ja sitä ja sen osia testattiin käytännössä. Ympäristön rakentamisen työvaiheet dokumentoitiin. Robotilla tuotettiin kartta-aineistoja, joita tallennettiin ympäristöön kuuluvaan HDFS-pilvipalveluun. Kartta-aineistot olivat myös ladattavissa robotin koneelle HDFS:stä. Robotin ohjausta varten suunniteltiin ja toteutettiin testikäyttöliittymä, jolla robottia voi ohjata joko robotin omalta koneelta tai etäkoneelta. Robotilla kyettiin navigoimaan kartoitetussa ympäristössä ja sen kartta-aineistoja voitiin manipuloida, robotin kulkuaan varten laskemien polkujen ohjailemiseksi. Robotti osasi myös kulkiessaan väistää eteensä tulevia esteitä. Ympäristössä käytettiin ainoastaan open source -ohjelmistoja, lukuun ottamatta työssä toteutettua testikäyttöliittymää. Käyttöliittymä tehtiin vain omaan käyttöön, mutta myös se voidaan tarvittaessa julkaista myöhemmin open source -jakeluna.

Ympäristön rakentamisen haasteet liittyivät muun muassa ROSin ja Hadoopin hajanaiseen dokumentaatioon. Tarvittavan tiedon löytäminen saattoi vaatia paljon työtä. Testikäyttöliittymän yhdistäminen ROSiin sujui varsin kivuttomasti, kiitos mahdollisuuden luoda ROSin työkaluilla valmis ROS-Qt -projektipohja. Sen avulla nähtiin, miten käyttöliittymä liittyy ROS-ympäristöön. Samalla päästiin tutustumaan Qt-ympäristöön, jota valitettavasti ei yrityksistä huolimatta saatu toimimaan täydellisesti. Kehitystyö Qt-kehyksen päälle onnistui kuitenkin hyvin ja yhdistämällä siihen GitLab, saatiin myös versionhallinta toteutettua sujuvasti.

Ympäristön toiminnan kannalta suurimmiksi haasteiksi osoittautuivat langaton tiedonsiirto WLANia käyttäen, sekä robottien 3D-sensorien rajallinen havaintokyky. Meidän ruuhkaisessa käyttöympäristössämme useamman tukiaseman käyttö osoittautui vaikeaksi. WLAN-tukiaseman vaihtaminen vei usein liikaa aikaa, jotta ROS-prosessit olisivat pysyneet sen aikana käynnissä. Sensorien osalta ongelmia aiheuttivat minimi- ja maksimimittausetäisyydet, sekä keilan rajallisuus.

WLAN-tukiaseman vaihtoa on syytä testata myös rauhallisemmassa ympäristössä. Samoin roaming aggressiveness -parametria tukevaa tietokonetta/päätelaitetta tulisi kokeilla parametrin vaikutuksen selvittämiseksi. Yhden tukiaseman konfiguraatiossa sen peittoa voidaan jatkaa käyttämällä robotin tietokoneessa WLAN-sovitinta, jossa on ulkoinen antenni. Mikäli WLAN-yhteyttä ei saada toimimaan koko robotin toiminta-alueella, voidaan harkita 3G/4G yhteyttä – tässä on muistettava, että vaatimuksena on kiinteä IP-osoite.

Sensorien minimimittausetäisyyttä voidaan kompensoida esimerkiksi liittämällä robotin puskuriin mikrokontrollerilla luettavia IR-etäisyys-sensoreita, jolloin robotin lähialueen tunnistus paranee. Aihetta on käsitelty Yujin Robotin sivuilla lähteessä [120], jossa on myös linkki hakemistoon, josta löytyy sensoreiden Turtlebotiin asentamista varten 3D-tulostettava kiinnitysrima.

Paras sensorivaihtoehto olisi ympäriskannaava laser. Käyttöympäristö määrää laserin riittävän maksimikantaman. Maksimikantaman tulisi mielellään olla niin pitkä, että robotti saa kulkiessaan aina sellaista mittaustietoa, jossa on muuttuva etäisyys johonkin kohteeseen. Tällaisia kohteita ovat esimerkiksi käytävällä olevat tolpat tai sen peräseinät. Samalla minimimittausetäisyyden tulisi olla samaa luokkaa, kuin lyhin etäisyys laserista Turtlebotin ulkoreunaan. Edellä mainituista IR-sensoreista olisi hyötyä myös ympäriskannaavalla laserilla mitattaessa, sillä tasossa skannaava laser ei sijoituspaikastaan riippuen välttämättä havaitse hyvin matalalla sijaitsevia kohteita, kuten ihmisen jalkaterää. Tällöin IR-antureilta saataisiin tärkeää lisäinformaatiota.

Monirobottiympäristöä toteutettaessa kannattaa siirtyä ROSin uudempaan ROS 2 -versioon. Se toteuttaa monen robotin ympäristön yksinkertaisemmin, kuin meidän nykyinen versiomme. ROS 2:n tilannekatsaus julkistettiin ROSCON 2016:ssa, joka pidettiin 8.–9.10.2016. Tilaisuuden videoidut esitykset ja materiaali tulevat Open Source Robotics Foundationin ilmoituksen mukaan saataville heidän sivuilleen 20.10.2016 mennessä [121].

ROS 2 elää tätä kirjoitettaessa voimakasta kehitysvaihetta – sen maturiteettia on syytä seurata. Vielä tässä vaiheessa kannattaa harkita nykyisen ympäristön toiminnan parantamista edellä esitetyillä keinoilla. ROS 2:een kannattanee siirtyä vasta, kun siitä julkaistaan ensimmäinen LTS versio.

LÄHTEET

[1] International Federation of Robotics, “Service Robot Statistics.” [Online]. Available: http://www.ifr.org/service-robots/statistics/. [Accessed: 06-Aug-2016].

[2] RoboEarth, “RoboEarth.” [Online]. Available: http://roboearth.org/. [Accessed: 19-Jun-2016].

[3] R. Arumugam, V. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. Kong, As. Kumar, K. Meng, and G. Jit, “DAvinCi: A cloud computing framework for service robots,” in *International Conference on Robotics and Automation*, 2010, pp. 3084–3089.

[4] KnowRob, “KnowRob: Knowledge processing for robots.” [Online]. Available: http://www.knowrob.org/knowrob. [Accessed: 03-Jul-2016].

[5] Rapuyta Robotics, “Rapuyta Robotics.” [Online]. Available: http://www.rapyuta-robotics.com/. [Accessed: 03-Jul-2016].

[6] ISO, “ISO 8373:2012(en), Robots and robotic devices — Vocabulary.” 2012.

[7] V. Baya and L. Wood, “Technology Forecast: Service robots - The next big productivity platform: PwC,” *PwC*, 2015. [Online]. Available: http://www.pwc.com/us/en/technology-forecast/2015/robotics/features/service-robots-big-productivity-platform.html. [Accessed: 03-Sep-2016].

[8] ISO, “ISO 13482:2014 - Robots and robotic devices -- Safety requirements for personal care robots.” 2014.

[9] F. Dayoub, T. Morris, and P. I. Corke, “Rubbing Shoulders With Mobile Service Robots.,” *IEEE Access*, vol. 3, pp. 333–342, 2015.

[10] ROS.org, “ROS/Introduction,” 2014. [Online]. Available: http://wiki.ros.org/ROS/Introduction. [Accessed: 19-Jun-2016].

[11] Opensource.org, “The BSD 3-Clause License.” [Online]. Available: https://opensource.org/licenses/BSD-3-Clause. [Accessed: 17-Jul-2016].

[12] ROS.org, “Is ROS For Me?” [Online]. Available: http://www.ros.org/is-ros-for-me/. [Accessed: 17-Jul-2016].

[13] J. M. O’Kane and J. M. O. Kane, *A gentle introduction to ROS*. O’Kane, 2013.

[14] ROS.org, “Client Libraries,” 2016. [Online]. Available: http://wiki.ros.org/Client Libraries. [Accessed: 09-Jul-2016].

[15] ROS.org, “IDEs,” 2016. [Online]. Available: http://wiki.ros.org/IDEs. [Accessed: 09-Jul-2016].

[16] Open Source Robotics Foundation, “Open Source Robotics Foundation.” [Online]. Available: http://www.osrfoundation.org/. [Accessed: 16-Aug-2016].

[17] A. Martinez and E. Fernández, *Learning ROS for Robotics Programming*. Packt Publishing, 2013.

[18] ROS.org, "Master," 2012. [Online]. Available: http://wiki.ros.org/Master. [Accessed: 09-Jul-2016].

[19] XML-RPC.Com, "XMLRPC." [Online]. Available: http://xmlrpc.scripting.com/. [Accessed: 09-Jul-2016].

[20] ROS.org, "Parameter Server," 2013. [Online]. Available: http://wiki.ros.org/Parameter Server. [Accessed: 19-Jun-2016].

[21] I. döt N. Oren Ben-Kiki, Clark Evans, "YAML Ain't Markup Language (YAML™) Version 1.2." [Online]. Available: http://yaml.org/spec/1.2/spec.html. [Accessed: 10-Jul-2016].

[22] K. Conley, "rosparam," 2014. [Online]. Available: http://wiki.ros.org/rosparam. [Accessed: 19-Jun-2016].

[23] T. Foote and R. B. Rusu, "nodelet," 2013. [Online]. Available: http://wiki.ros.org/nodelet. [Accessed: 09-Jul-2016].

[24] ROS.org, "Nodes," 2012. [Online]. Available: http://wiki.ros.org/Nodes. [Accessed: 09-Jul-2016].

[25] ROS.org, "ROSConcepts," 2014. [Online]. Available: http://wiki.ros.org/ROS/Concepts. [Accessed: 10-Jul-2016].

[26] ROS.org, "ROS/TCPROS," 2013. [Online]. Available: http://wiki.ros.org/ROS/TCPROS. [Accessed: 10-Jul-2016].

[27] ROS.org, "ROS/UDPROS," 2013. [Online]. Available: http://wiki.ros.org/ROS/UDPROS. [Accessed: 10-Jul-2016].

[28] ROS.org, "Topics," 2014. [Online]. Available: http://wiki.ros.org/Topics. [Accessed: 10-Jul-2016].

[29] ROS.org, "rostopic," 2016. [Online]. Available: http://wiki.ros.org/rostopic. [Accessed: 10-Jul-2016].

[30] K. Conley and T. Foote, "rosmsg," 2016. [Online]. Available: http://wiki.ros.org/rosmsg. [Accessed: 16-Jul-2016].

[31] "ROS/Connection Header," 2011. [Online]. Available: http://wiki.ros.org/ROS/Connection Header. [Accessed: 08-Aug-2016].

[32] ROS.org, "Messages," 2015. [Online]. Available: http://wiki.ros.org/Messages. [Accessed: 16-Jul-2016].

[33] ROS.org, "Services," 2012. [Online]. Available: http://wiki.ros.org/Services. [Accessed: 16-Jul-2016].

[34] ROS.org, “srv,” 2013. [Online]. Available: http://wiki.ros.org/srv. [Accessed: 16-Jul-2016].

[35] ROS.org, “ROS/Technical Overview,” 2014. [Online]. Available: http://wiki.ros.org/ROS/Technical Overview#Establishing_a_service_connection. [Accessed: 16-Jul-2016].

[36] E. Marder-Eppstein, “Navigation,” 2016. [Online]. Available: http://wiki.ros.org/navigation. [Accessed: 20-Aug-2016].

[37] D. Thomas, “Changes between ROS 1 and ROS 2.” [Online]. Available: http://design.ros2.org/articles/changes.html. [Accessed: 21-Aug-2016].

[38] D. Thomas, “ROS 2 middleware interface.” [Online]. Available: http://design.ros2.org/articles/ros_middleware_interface.html. [Accessed: 21-Aug-2016].

[39] A. Tiderko, “multimaster_fkie,” 2015. [Online]. Available: http://wiki.ros.org/multimaster_fkie. [Accessed: 05-Aug-2016].

[40] Open Source Robotics Foundation, “ROS on DDS.” [Online]. Available: http://design.ros2.org/articles/ros_on_dds.html. [Accessed: 21-Aug-2016].

[41] OMG, “The Real-time Publish-Subscribe (RTPS) Wire Protocol DDS Interoperability Wire Protocol Specification v2.2,” vol. 15, no. September, 2014.

[42] M. Quigley, “ROS2 on ‘small’ embedded systems,” 2015, p. 17.

[43] D. Thomas, “Alpha7 Overview,” 2016. [Online]. Available: https://github.com/ros2/ros2/wiki/Alpha7-Overview. [Accessed: 05-Aug-2016].

[44] D. Thomas, “Migration Guide,” 2016. [Online]. Available: https://github.com/ros2/ros2/wiki/Migration-Guide. [Accessed: 22-Aug-2016].

[45] TutorialsPoint, “Hadoop - Big Data Overview,” 2016. [Online]. Available: http://www.tutorialspoint.com/hadoop/hadoop_big_data_overview.htm. [Accessed: 19-Jul-2016].

[46] The Apache Software Foundation, “Apache License 2.0,” 2004. [Online]. Available: http://www.apache.org/licenses/LICENSE-2.0. [Accessed: 17-Jul-2016].

[47] T. White, *Hadoop : The Definitive Guide*, 3rd ed. O’Reilly, 2012.

[48] Apache.org, “MapReduce Tutorial,” 2016. [Online]. Available: https://hadoop.apache.org/docs/r2.7.2/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. [Accessed: 20-Jul-2016].

[49] A4Academics, “Hadoop Map Reduce Architecture and Example,” 2014. [Online]. Available: http://a4academics.com/tutorials/83-hadoop/840-map-reduce-architecture. [Accessed: 20-Jul-2016].

[50] The Apache Software Foundation, “HDFS Architecture,” 2016. [Online]. Available: http://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html. [Accessed: 24-Jul-2016].

[51] The Qt Company, “The Qt Company,” 2016. [Online]. Available: https://www.qt.io/company/. [Accessed: 12-Aug-2016].

[52] The Qt Company Ltd, “Licensing,” *2016*. [Online]. Available: https://www.qt.io/licensing/. [Accessed: 27-Jul-2016].

[53] The Qt Company Ltd, “About Qt,” 2016. [Online]. Available: http://wiki.qt.io/About_Qt. [Accessed: 27-Jul-2016].

[54] GitHub Inc., “Learn Git and GitHub without any code!,” 2016. [Online]. Available: https://github.com/. [Accessed: 28-Jul-2016].

[55] GitLab.com, “GitLab.” [Online]. Available: https://about.gitlab.com/. [Accessed: 05-Sep-2016].

[56] GitLab.com, “GitLab Documentation/Requirements.” [Online]. Available: http://docs.gitlab.com/ce/install/requirements.html#non-unix-operating-systems-such-as-windows. [Accessed: 28-Jul-2016].

[57] GitLab.com, “Download GitLab Community Edition.” [Online]. Available: https://about.gitlab.com/downloads/#ubuntu1604. [Accessed: 28-Jul-2016].

[58] iRobot, “History | iRobot,” 2016. [Online]. Available: http://www.irobot.com/About-iRobot/Company-Information/History.aspx. [Accessed: 03-Sep-2016].

[59] iRobot, “Create 2 Programmable Robot | iRobot,” 2016. [Online]. Available: http://www.irobot.com/About-iRobot/STEM/Create-2.aspx. [Accessed: 03-Sep-2016].

[60] iRobot, “iRobot ® Create Owner’s guide Important Safety Instructions,” 2006.

[61] Rose Hulman Institute of Technology, “Getting to know the iRobot Create,” 2008.

[62] Robotnik, “Yujin Robot | Robotnik.” [Online]. Available: http://www.robotnik.eu/yujin-robot/. [Accessed: 03-Sep-2016].

[63] IEEE-Spectrum, “TurtleBot 2 Prototype Unveiled at ROSCon - IEEE Spectrum,” 2012. [Online]. Available: http://spectrum.ieee.org/automaton/robotics/diy/turtlebot-2-prototypes-unveiled-at-roscon-2. [Accessed: 03-Sep-2016].

[64] Robotnik Automation, “Turtlebot II, User’s Manual (Indigo v1).” .

[65] Microsoft, “‘Kinect for Xbox 360’ is Official Name of Microsoft’s Controller-Free Game Device | News Center,” 2010. [Online]. Available: https://news.microsoft.com/2010/06/13/kinect-for-xbox-360-is-official-name-of-microsofts-controller-free-game-device/#sm.0000i2gb4iz5kdv6qug1uvw6zze6b.

[Accessed: 03-Sep-2016].

[66] Microsoft, “Coordinate Spaces,” 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/hh973078.aspx#Depth_Ranges. [Accessed: 11-Oct-2016].

[67] OpenKinect.org, “OpenKinect Protocol Documentation.” [Online]. Available: https://openkinect.org/wiki/Protocol_Documentation#Control_Commands;a=summary. [Accessed: 18-Jun-2016].

[68] Robotnik Store, “Kinect Xbox 360.” [Online]. Available: http://www.robotnikstore.com/robotnik/5121637/kinect-xbox-360.html. [Accessed: 18-Jun-2016].

[69] M. R. Andersen, T. Jensen, P. Lisouski, A. K. Mortensen, M. K. Hansen, T. Gregersen, and P. Ahrendt, “Kinect Depth Sensor Evaluation for Computer Vision Applications,” *Electr. Comput. Eng.*, vol. Technical, 2012.

[70] J. Howse, S. Puttemans, Q. Hua, and U. Sinha, *OpenCV 3 Blueprints*, 1st ed. Birmingham: Packt Publishing, 2015.

[71] ASUSTeK Computer Inc., “Xtion PRO LIVE.” [Online]. Available: http://www.asus.com/3D-Sensor/Xtion_PRO_LIVE/specifications/. [Accessed: 29-Jun-2016].

[72] T. Bernhard, A. Chintalapally, and D. Zukowski, “A Comparative Study of Structured Light and Laser Range Finding Sensors,” 2012.

[73] BusinessWire, “Orbbec Unveils World’s First 3D Camera-Computer | Business Wire,” 2015. [Online]. Available: http://www.businesswire.com/news/home/20151001005621/en/Orbbec-Unveils-World%E2%80%99s-3D-Camera-Computer. [Accessed: 13-Sep-2016].

[74] Orbbec, “Orbbec Persee: World’s First 3D Camera-Computer | Indiegogo,” 2016. [Online]. Available: https://www.indiegogo.com/projects/orbbec-persee-world-s-first-3d-camera-computer#/. [Accessed: 05-Sep-2016].

[75] Orbbec, “Astra & Astra S.” [Online]. Available: https://orbbec3d.com/product-astra/. [Accessed: 18-Jun-2016].

[76] Orbbec, “Astra Pro | Orbbec,” 2016. [Online]. Available: https://orbbec3d.com/product-astra-pro/. [Accessed: 13-Sep-2016].

[77] Orbbec, “Persee | Orbbec,” 2016. [Online]. Available: https://orbbec3d.com/product-persee/. [Accessed: 13-Sep-2016].

[78] L. TP-LINK Technologies Co., “Archer C2600 Overview,” 2016. [Online]. Available: http://www.tp-link.com/en/products/details/Archer-C2600.html#overview. [Accessed: 28-Jul-2016].

[79] L. TP-LINK Technologies Co., “Archer C2600 Specifications.” [Online]. Available: http://www.tp-link.fi/products/details/cat-9_Archer-

C2600.html#specifications. [Accessed: 28-Jul-2016].

[80] TP-Link, "MU-MIMO." [Online]. Available: http://www.tp-link.com/common/promo/en/MU-MIMO/MU-MIMO.html. [Accessed: 13-Aug-2016].

[81] J. Hintersteiner, "How Does MU-MIMO Work? | Network Computing," *Information Week/Network Computing*, 2016.

[82] Cisco, "802.11ac: The Fifth Generation of Wi-Fi," 2014. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/wireless/aironet-3600-series/white_paper_c11-713103.html. [Accessed: 13-Aug-2016].

[83] O. Aoki, "Debian Reference," *Architecture*, 2007. [Online]. Available: http://qref.sourceforge.net/quick/ch-gateway.en.html. [Accessed: 14-Aug-2016].

[84] Debian.org, "Network setup," 2016. [Online]. Available: http://www.debian.org/doc/manuals/debian-reference/ch05.en.html#_the_hostname_resolution. [Accessed: 16-Aug-2016].

[85] ROS.org, "ROS/Tutorials/catkin/CreatingPackage - ROS Wiki," 2016. [Online]. Available: http://wiki.ros.org/ROS/Tutorials/catkin/CreatingPackage. [Accessed: 13-Oct-2016].

[86] M. Silliman, "Installing ROS." [Online]. Available: http://learn.turtlebot.com/2015/02/01/5/. [Accessed: 16-Aug-2016].

[87] ROS.org, "Turtlebot Installation," 2015. [Online]. Available: http://wiki.ros.org/turtlebot/Tutorials/indigo/Turtlebot Installation. [Accessed: 16-Aug-2016].

[88] ROS.org, "Overlaying with catkin workspaces," 2014. [Online]. Available: http://wiki.ros.org/catkin/Tutorials/workspace_overlaying. [Accessed: 20-Aug-2016].

[89] thinkwiki.org, "Power Connector - ThinkWiki," 2016. [Online]. Available: http://www.thinkwiki.org/wiki/Power_Connector. [Accessed: 27-Aug-2016].

[90] "Running Hadoop On Ubuntu Linux (Multi-Node Cluster) - Michael G. Noll." [Online]. Available: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/. [Accessed: 30-May-2016].

[91] K. Hong, "Hadoop 2.6 - Installing on Ubuntu 14.04 (Single-Node Cluster) - 2016," 2016. [Online]. Available: http://www.bogotobogo.com/Hadoop/BigData_hadoop_Install_on_ubuntu_single_node_cluster.php. [Accessed: 25-Sep-2016].

[92] HadoopWorld, *How to install hadoop 2.7.1 single node cluster ubuntu 14.04 - YouTube*. YouTube, 2015.

[93] Apache, "HadoopJavaVersions - Hadoop Wiki," 2016. [Online]. Available: http://wiki.apache.org/hadoop/HadoopJavaVersions. [Accessed: 14-Sep-2016].

[94] S. Chawla, “Install a Multi Node Hadoop Cluster on Ubuntu 14.04 – Sumit Chawla’s Blog,” 2015. [Online]. Available: https://chawlasumit.wordpress.com/2015/03/09/install-a-multi-node-hadoop-cluster-on-ubuntu-14-04/. [Accessed: 25-Sep-2016].

[95] PC Magazine Encyclopedia, “Hyperthreading Definition.” [Online]. Available: http://www.pcmag.com/encyclopedia/term/44567/hyperthreading. [Accessed: 29-Sep-2016].

[96] TechSpot, “inSSIDer 3.1.2.1,” 2016. [Online]. Available: http://www.techspot.com/downloads/5936-inssider.html. [Accessed: 22-Aug-2016].

[97] I. Sommerville, *Software Engineering*, 9th ed. Pearson, 2010.

[98] ROS.org, “rviz - ROS Wiki,” 2016. [Online]. Available: http://wiki.ros.org/rviz. [Accessed: 18-Sep-2016].

[99] The Qt Company, “Download Qt Open Source.” [Online]. Available: https://www.qt.io/download-open-source/. [Accessed: 31-Jul-2016].

[100] ROS.org, “Qt App Templates,” 2016. [Online]. Available: http://wiki.ros.org/qt_create/Tutorials/Qt App Templates. [Accessed: 27-Jul-2016].

[101] ROS.org, “Building and using catkin packages in a workspace,” 2015. [Online]. Available: http://wiki.ros.org/catkin/Tutorials/using_a_workspace. [Accessed: 20-Aug-2016].

[102] S. Sijbrandij, “GitLab Flow,” 2014. [Online]. Available: https://about.gitlab.com/2014/09/29/gitlab-flow/. [Accessed: 02-Aug-2016].

[103] Robert Bosch GmbH, “Bosch PLR 15 Digital Laser Measure.” [Online]. Available: http://www.bosch-plr15.com/gb/en/technical-data.html. [Accessed: 17-Sep-2016].

[104] E. J. Olaya, F. Berry, Y. Mezouar, I. Pascal, and U. B. P. U. M. R. Cnrs, “A Robotic Structured Light Camera,” *Conf. Publ. 2014 IEEE/ASME Int. Conf. Adv. Intell. Mechatronics*, pp. 727–734, 2014.

[105] B. Freedman, A. Sphunt, M. Macline, and Y. Arieli, “Depth mapping using projected patterns,” US20100118123, 2008.

[106] A. Sphunt, “Depth mapping using multi-beam illumination,” US20100020078, 2008.

[107] P. Zanuttigh, G. Marin, C. D. Mutto, F. Dominio, L. Minto, and G. M. Cortelazzo, *Time-of-Flight and Structured Light Depth Cameras - Technology and Applications*. Springer, 2016.

[108] B. Gerkey, “Gmapping,” 2015. [Online]. Available: http://wiki.ros.org/gmapping. [Accessed: 28-Aug-2016].

[109] G. Grisetti, C. Stachniss, and W. Burgard, “OpenSLAM.org.” [Online]. Available: http://openslam.org/gmapping.html. [Accessed: 28-Aug-2016].

[110] G. Grisetti, C. Stachniss, and W. Burgard, “Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2005, no. January, pp. 2432–2437, 2005.

[111] G. Grisetti, C. Stachniss, and W. Burgard, “Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters,” *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 34–46, 2007.

[112] K. P. Murphy, “Bayesian Map Learning in Dynamic Environments,” *Neural Info. Proc. Syst.*, 1999.

[113] M. W. Silliman, “Teleoperation,” 2015. [Online]. Available: http://learn.turtlebot.com/2015/02/01/9/. [Accessed: 27-Aug-2016].

[114] ROS.org, “Gmapping inconsistent between identical runs - ROS Answers: Open Source Q&A Forum,” 2014. [Online]. Available: http://answers.ros.org/question/159426/gmapping-inconsistent-between-identical-runs/. [Accessed: 18-Sep-2016].

[115] B. Gerkey, “gmapping: slam_gmapping.cpp Source File.” [Online]. Available: http://docs.ros.org/indigo/api/gmapping/html/slam__gmapping_8cpp_source.html. [Accessed: 20-Sep-2016].

[116] Gimp.org, “GIMP - GNU Image Manipulation Program,” 2016. [Online]. Available: https://www.gimp.org/. [Accessed: 27-Aug-2016].

[117] ROS.org, “navigation/Tutorials/RobotSetup - ROS Wiki,” 2015. [Online]. Available: http://wiki.ros.org/navigation/Tutorials/RobotSetup#Costmap_Configuration_.28local_costmap.29_.26_.28global_costmap.29. [Accessed: 02-Oct-2016].

[118] E. Marder-Eppstein, D. V. Lu, and D. Hershberger, “costmap_2d - ROS Wiki,” 2015. [Online]. Available: http://wiki.ros.org/costmap_2d. [Accessed: 02-Oct-2016].

[119] Wireshark.org, “Wireshark · Go Deep.,” 2016. [Online]. Available: https://www.wireshark.org/. [Accessed: 04-Oct-2016].

[120] Yujinrobot, “IR-Sensors,” 2015. [Online]. Available: http://kobuki.yujinrobot.com/wiki/ir-sensors/. [Accessed: 14-Oct-2016].

[121] Open Source Robotics Foundation, “ROSCon 2016,” 2016. [Online]. Available: http://roscon.ros.org/2016/. [Accessed: 14-Oct-2016].

LIITE A: YHDEN KONEEN HADOOP-KLUSTERIN ASENNUS

Suositellun Java-version asennus:

```
// poistettiin koneessa ollut Open JDK
sudo apt-get purge openjdk*
// asennettiin lähteidenhallintatyökalu
sudo apt-get install software-properties-common
// lisättiin PPA (Personal Packet Archive) lähdeluetteloon
sudo add-apt-repository ppa:webupd8team/java
// päivitettiin pakettiluettelo
sudo apt-get update
// asennettiin uusin versio Oraclen Java 7:stä (1.7.0_80)
sudo apt-get install oracle-java7-installer
```

Hadoop-käyttäjäryhmän ja käyttäjän luominen:

```
sudo addgroup hadoop                    // lisättiin ryhmä hadoop
sudo adduser --ingroup hadoop user      // lisättiin ryhmään user
sudo adduser user sudo                  // lisättiin user sudoeriksi
```

Rsync-asennus (Remote Sync):

```
sudo apt-get rsync
```

SSH:n (Secure Shell) asennus ja salasanattoman yhteydenmuodostuksen käyttöönotto:

```
sudo apt-get install openssh-server     // asennettiin OpenSSH-palvelin
su user                                 // kirjauduttiin sisään userina
cd                                      // siirryttiin userin kotihakemistoon
// luotiin RSA-avainpari ilman salasanaa
ssh-keygen -t rsa -P ""
// lisättiin yleinen avain luotettujen listaan, jotta ssh-yhteys localhostiin onnistuu
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
// kokeiltiin ssh:ta, pitää mennä ilman salasanaa
ssh localhost
exit                                    // OK, suljettiin ssh-yhteys
```

Hadoop-tiedostojen kopiointi koneelle:

```
wget http://mirrors.sonic.net/apache/hadoop/common/hadoop-2.7.2/hadoop-2.7.2.tar.gz
                                        // haettiin Hadoop-paketti
tar xvzf hadoop-2.7.2.tar.gz            // purettiin se
sudo mv hadoop-2.7.2 /usr/local/hadoop  // siirrettiin sopivaan sijaintiin
// asetettiin user ko. hakemiston omistajaksi
sudo chown -R user /usr/local/hadoop
```

Komentotulkin alustustiedostoon (~/.bashrc) on lisättävä Hadoop-ympäristömuuttujat:

```
// lisättiin Hadoopin sijaintimääritykset
export JAVA_HOME=/usr/lib/jvm/java-7-oracle
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
// lisättiin bin- ja sbin-hakemistot polkuihin
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
```

Ajettiin .bashrc, jotta muutokset tulivat voimaan

```
source ~/.bashrc
```

Seuraavaksi konfiguroitiin Hadoopin määrittelytiedostot, jotka löytyvät polusta /usr/local/hadoop/etc/hadoop/. Muutettavat tai lisättävät kohdat on seuraavassa lihavoitu.

Asetettiin Javan kotihakemisto *hadoop-env.sh*:hon:

```
// Javan kotihakemisto
export JAVA_HOME=/usr/lib/jvm/java-7-oracle
```

Lisättiin oletustietojärjestelmän nimi *core-site.xml*:ään:

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://192.168.1.100:9000</value>  // nimi
  </property>
</configuration>
```

Lisättiin replikointiarvo eli datablokkien kopioiden oletusmäärä ja tallennuspolut nimitaululle ja datablokeille *hdfs-site.xml*:ään:

```
<configuration>
  <property>
    <name>dfs.replication</name>                  // replikointiarvo
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>    // nimitaulun tallennuspolku
```

```
      <value>file:/usr/local/hadoop_tmp/hdfs/namenode</value>
    </property>
    <property>
      <name>dfs.datanode.data.dir</name>       // datablokkien tallennuspolku
      <value>file:/usr/local/hadoop_tmp/hdfs/datanode</value>
    </property>
</configuration>
```

Konfiguroitiin Map-Reducen shuffle-palvelu *yarn-site.xml*:ssä:

```
<configuration>
   <property>
      <name>yarn.nodemanager.aux-services</name>
       <value>mapreduce_shuffle</value>// lisäpalvelun nimi
   </property>
   <property>
      <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
      // palvelun toteuttava luokka
      <value>org.apache.hadoop.mapred.ShuffleHandler</value>
   </property>
</configuration>
```

Kopioitiin *mapred-site.xml.template* pohjaksi *mapred-site.xml*:lle:

```
cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template
/usr/local/hadoop/etc/hadoop/mapred-site.xml
```

Lisättiin *mapred-site.xml*:ään määritys, jolla kerrotaan käytettävän MapReduce v2:sta (YARN):

```
<configuration>
   <property>
      <name>mapreduce.framework.name</name>
      <value>yarn</value>                    // MapReduce v2 käyttää
   </property>
</configuration>
```

Luotiin Hadoopin työhakemistot:

```
sudo mkdir -p /usr/local/hadoop_tmp
sudo mkdir -p /usr/local/hadoop_tmp/hdfs/namenode
sudo mkdir -p /usr/local/hadoop_tmp/hdfs/datanode
```

Asetettiin user omistajaksi edellä luotuihin hakemistoihin:

```
sudo chown -R user /usr/local/hadoop_tmp
```

Alustettiin HDFS:n tiedostojärjestelmä:

```
hdfs namenode -format
```

LIITE B: KAHDEN KONEEN HADOOP-KLUSTERIN JA SEN CLIENTIN KONFIGUROINTI

Muutettiin kaikkien kolmen koneen core-site.xml:ään masterin IP-osoitteen (192.168.1.100) tilalle koneen hostname Hdmaster. Hdmasteriin oli kuitenkin jätettävä IP-osoite, koska muuten konfiguraatio ei toiminut.

Kaikkien koneiden *hdfs-site.xml* -tiedostoon muutettiin replikointiarvo kahdeksi, jotta tallennetuista tiedostoista saadaan oletuksena kopio molemmille koneille:

```
<property>
  <name>dfs.replication</name>
  <value>2</value>
</property>
```

Hdmasterin *hdfs-site.xml*:ssä poistettiin lisäksi käyttöoikeuksien tarkastaminen käytöstä, jotta vältyimme ACL:n (Access Control List) käyttöönotolta:

```
<property>
  <name>dfs.permissions.enabled</name>
  <value>false</value>
</property>
```

Kaikkien koneiden *yarn-site.xml*:ään lisättiin YARN-resurssienhallinnan osoitteet ja portit:

```
</property>
  <name>yarn.resourcemanager.scheduler.address</name>
  <value>Hdmaster:8030</value>
</property>
<property>
  <name>yarn.resourcemanager.address</name>
  <value>Hdmaster:8032</value>
</property>
<property>
   <name>yarn.resourcemanager.webapp.address</name>
   <value>Hdmaster:8088</value>
</property>
<property>
   <name>yarn.resourcemanager.resource-tracker.address</name>
   <value>Hdmaster:8031</value>
</property>
<property>
   <name>yarn.resourcemanager.admin.address</name>
   <value>Hdmaster:8033</value>
</property>
```

Hdmasterin ja Hdslaven /usr/local/hadoop/etc/hadoop/slaves-tiedostosta poistettiin localhost ja lisättiin koneet, joissa on DataNode ja NodeManager:

```
Hdmaster      // myös masterissa on edellä mainitut tässä kokonpanossa
Hdslave
```

Jotta klusterin ID olisi sama koko konfiguraatiossa, kopioitiin Hdmasterin datanoden clusterID-parametrin arvo /usr/local/hadoop_tmp/hdfs/datanode/current/VERSION -tiedostosta ja formatoitiin namenode tällä ID:llä:

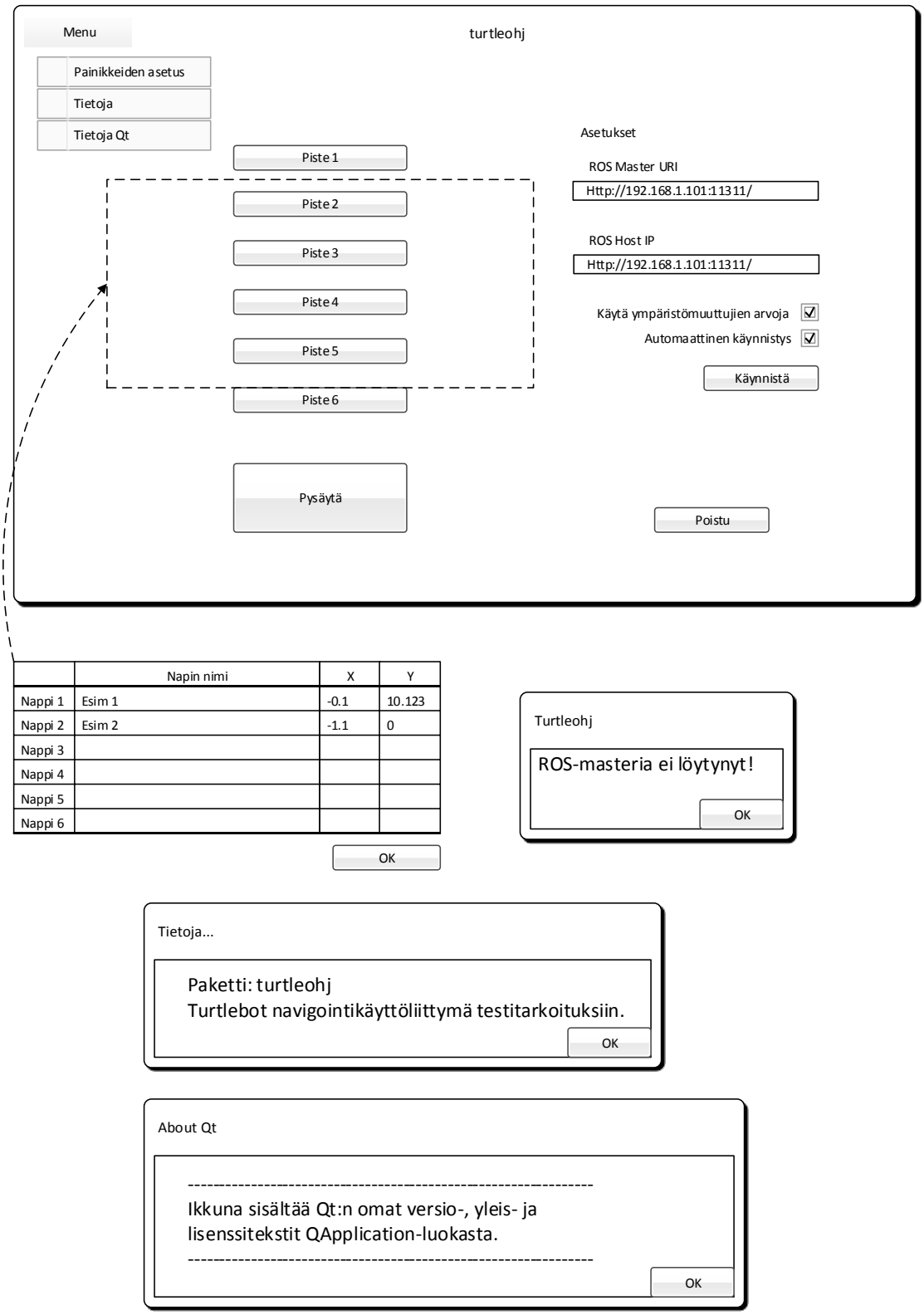
```
hdfs namenode -format -clusterId CID-199af6a7-74e1-40ac-825a-45577f09c98f
```

Kopioitiin vielä sama ID Hdslaven /usr/local/hadoop_tmp/hdfs/datanode/current/VERSION- ja /usr/local/hadoop_tmp/hdfs/namenode/current/VERSION -tiedostoihin.

Lisättiin vielä SSH:n käyttäjätunnuksien kyselyjen välttämiseksi user@Hdmaster:in julkinen SSH-avain tiedostosta $HOME/.ssh/id_rsa.pub käyttäjän user@Hdslave $HOME/.ssh/authorized_keys -tiedostoon:

```
user@Hdmaster:~$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub user@Hdslave
```

LIITE C: TURTLEOHJ-SIJOITTELUSUUNNITELMA



LIITE D: TURTLEOHJ-TESTIPÖYTÄKIRJA

Käynnistetään robotti ja ajetaan robotin koneessa omissa terminaaleissaan seuraavat komennot samassa järjestyksessä:

roslaunch turtlebot_bringup minimal.launch

roslaunch turtlebot_navigation amcl_demo.launch map_file:=/maps/my_map.yaml
(my_map.yaml = sopiva kartta)

Eäkoneessa käynnistetään Rviz koordinaattien määritystä varten:

roslaunch turtlebot_rviz_launchers view_navigation.launch

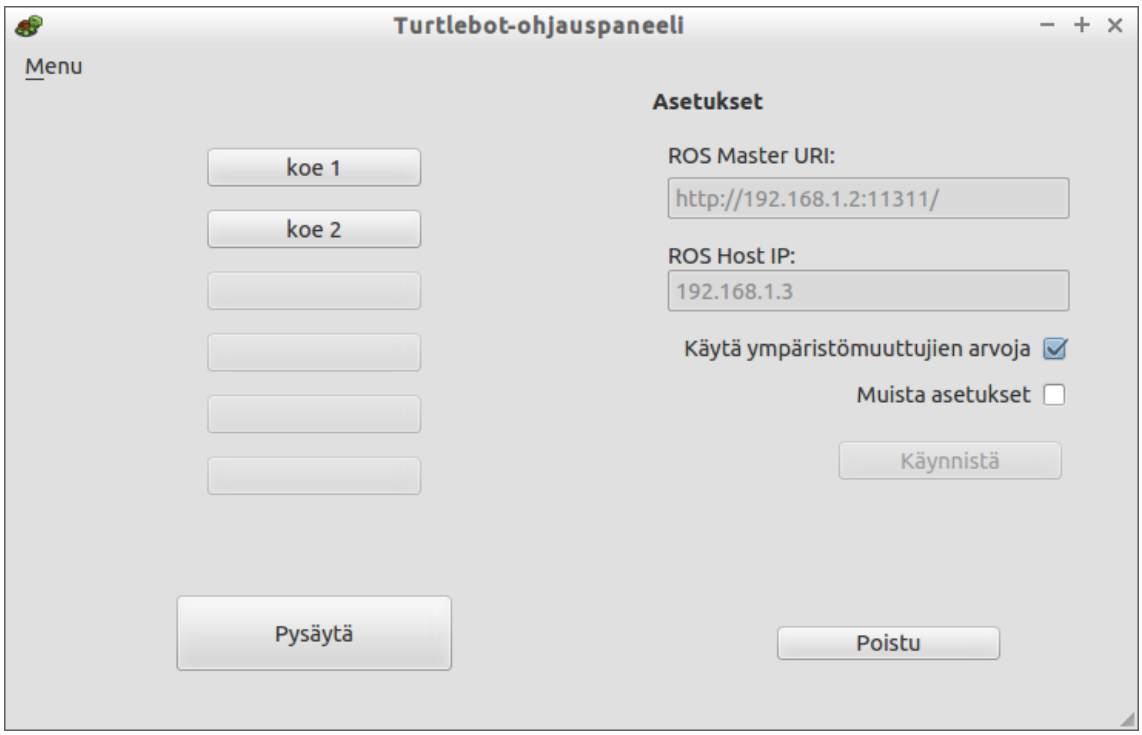
Painetaan rvizissä painiketta ”Publish Point”, jolloin kursorin sijainti näkyy ikkunan alareunassa koordinaatteina. Näistä vain kahta ensimmäistä käytetään (x, y).

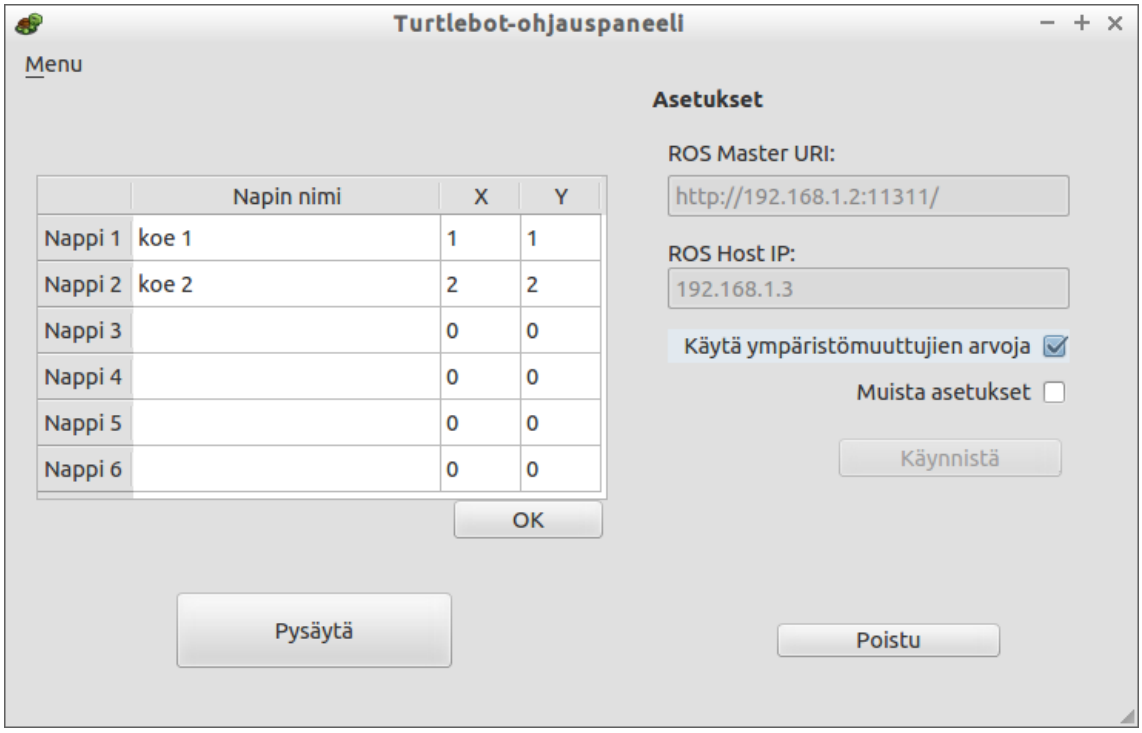
Käynnistetään turtleohj aluksi etäkoneessa, jolloin testaaminen helpompaa.

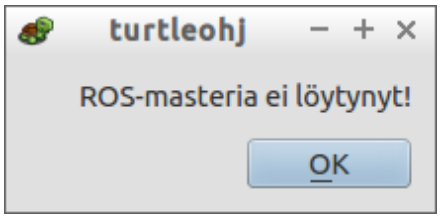
testinro / vaatimusnrot	Testi	Toiminta	Status	Huom.
1 / 1, 10, 11	annetaan master URI ja host-IP ja painetaan Käynnistys	navigointinapit ja ”Käynnistä” painike eivät valittavissa	OK	
2 / 7, 8	Valitaan Menu/Painikkeiden asetus	asetustaulukko aukeaa nappien tilalle	OK	
3 / 7, 8, 11	Syötetään Nappi 1:lle nimi sekä x- ja y-koordinaatit, painetaan OK	Nappi 1:n nimi näkyy ja nappi on valittavissa (muiden nav. nappien käyttö vielä estetty)	OK	
4 / 6, 8	Painetaan edellä konfiguroitua nappia	robotti lähtee liikkeelle n. 2s napin painamisen jälkeen ja ajaa annettuun kohteeseen	OK	
5 / 6, 7, 8, 11	Lisätään toinen nappi koordinaatteineen	molemmat nimetyt napit valittavissa, muut nav. napit eivät	OK	
6 / 6, 8	Painetaan edellä syötettyä nappia	robotti lähtee liikkeelle n. 2s napin painamisen jälkeen ja ajaa annettuun kohteeseen	OK	
7 / 6, 8	Painetaan uudelleen ensimmäisenä luotua nappia	robotti lähtee liikkeelle n. 2s napin painamisen jälkeen ja ajaa annettuun kohteeseen	OK	
8 / 7, 11	Poistetaan edellä painetun napin nimi painikkeiden asetuksen kautta	vain jäljelle jäävä valittavissa	OK	
9 / 6, 8, 11	Painetaan jäljelle jäänyttä nappia	robotti lähtee liikkeelle n. 2s napin painamisen jälkeen ja ajaa annettuun kohteeseen	OK	

10 / 12	Painetaan Poistu	ohjelma sammuu	OK	
11 / 5, 9	Käynnistetään turtleohj	IP:t ovat edellä asetetut, kaikki nav. napit estetty	OK	
12 / 11	Painetaan käynnistä	jäljellä oleva nappi on valittavissa	OK	
13 / 6, 7, 8	Annetaan nimi napille, josta se aiemmin poistettiin, painetaan sitä	robotti lähtee liikkeelle n. 2s napin painamisen jälkeen ja ajaa annettuun kohteeseen	OK	
14 / 2	Turtleohj uudelleenkäynnistys, editoidaan master URI http://1.1.1.1:1131/ host IP 1.1.1.1, valitaan käytä ympäristömuuttujien arvoja, uudelleenkäynnistys, painetaan käynnistys-nappia	ei virheilmoitusta masterin puuttumisesta	OK	
15 / 3	Turtleohj uudelleen käynnistys, poistetaan käytä ympäristömuuttujen arvoja, uudelleenkäynnistys, taulussa master URI http://1.1.1.1:1131/ host IP 1.1.1.1, käynnistä	virheilmoitus, masteria ei löydy	OK	Ilmoituksen tulo kestää < 2 min. Jos IP-osoite on oman verkon osoiteavaruudesta, ilmoitus tulee lähes heti.
16 / 4	Turtleohj uudelleenkäynnistys, anna oikeat master URI ja host IP ja valitse muista asetukset, sammuta ja käynnistä turtleohj	navigointi- poistu- ja pysäytysnapit valittavissa, käynnistysnappi ei	OK	
17 / 6, 13	lähetä robotti ao. nappia painamalla toiseen kohteeseen, paina Pysäytys-nappia, kun robotti on lähtenyt liikkeelle	robotti lähtee annettuun kohteeseen n. 2s napin painamisen jälkeen ja pysähtyy, kun Pysäytys-nappia painetaan	OK	
18 / 2, 6, 7, 8, 14	käynnistä toinenTurtleohj-instanssi robotin päällä olevaan koneeseen (rosrun turtleohj turtleohj __name:=turtleohj2) aseta ympäristömuuttujien asetukset käyttöön, määritä jokin sama piste ja aja robotti siihen	Turtleohj käynnistyy, eikä etäkoneen instanssi sammu alta. Robotti ajaa annettuun pisteeseen.	OK	
19 / 6, 14	Aja etäkoneelta toiseen pisteeseen.	robotti lähtee liikkeelle n. 2s napin painamisen jälkeen ja ajaa annettuun kohteeseen	OK	

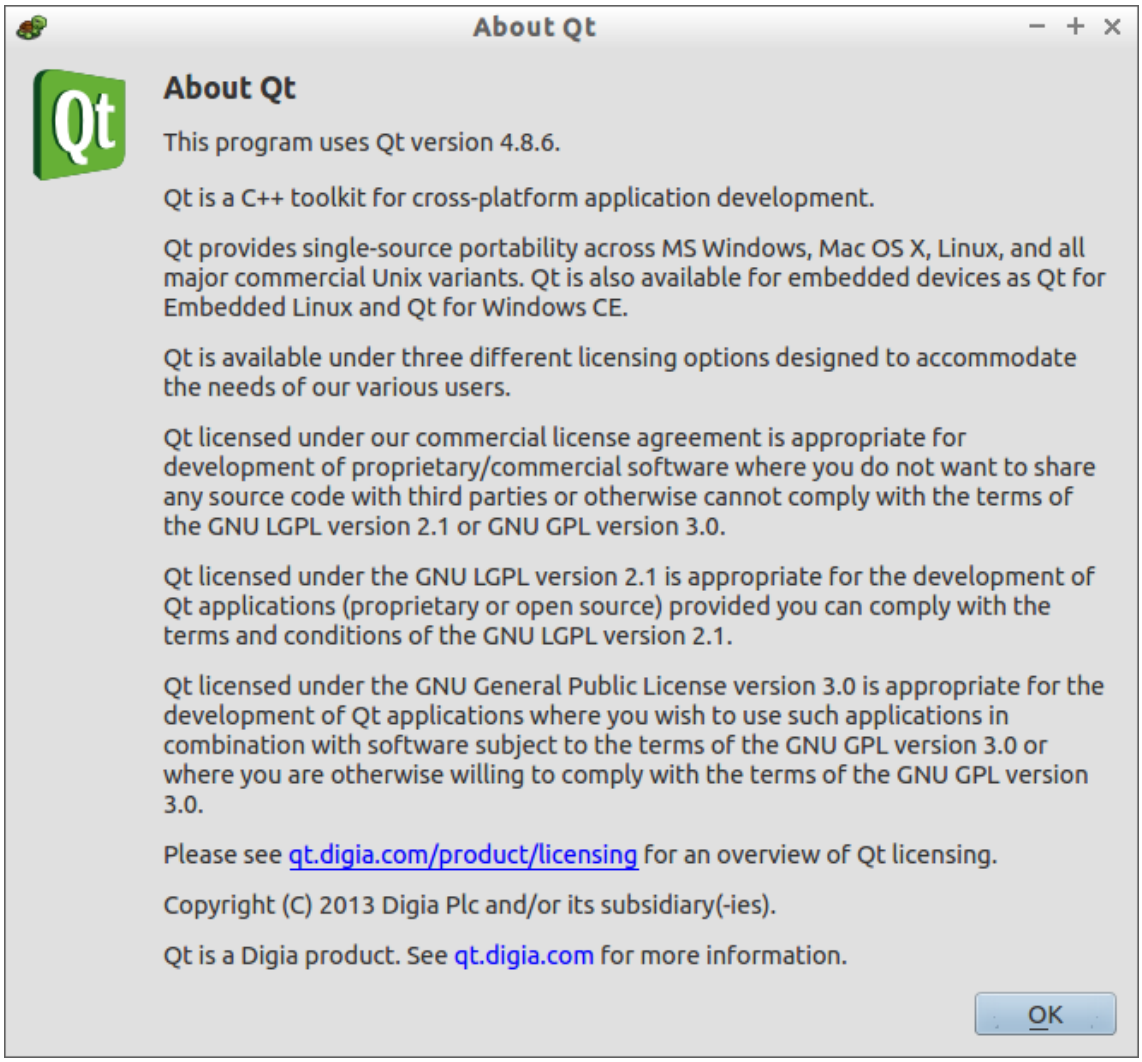
LIITE E: TURTLEOHJ-TOTEUTUS






turtleohj
ROS-masteria ei löytynyt!
OK


Tietoja...
Paketti: turtleohj
Turtlebot navigointikäyttöliittymä testitarkoituksiin.
OK


About Qt
About Qt
This program uses Qt version 4.8.6.
Qt is a C++ toolkit for cross-platform application development.
Qt provides single-source portability across MS Windows, Mac OS X, Linux, and all major commercial Unix variants. Qt is also available for embedded devices as Qt for Embedded Linux and Qt for Windows CE.
Qt is available under three different licensing options designed to accommodate the needs of our various users.
Qt licensed under our commercial license agreement is appropriate for development of proprietary/commercial software where you do not want to share any source code with third parties or otherwise cannot comply with the terms of the GNU LGPL version 2.1 or GNU GPL version 3.0.
Qt licensed under the GNU LGPL version 2.1 is appropriate for the development of Qt applications (proprietary or open source) provided you can comply with the terms and conditions of the GNU LGPL version 2.1.
Qt licensed under the GNU General Public License version 3.0 is appropriate for the development of Qt applications where you wish to use such applications in combination with software subject to the terms of the GNU GPL version 3.0 or where you are otherwise willing to comply with the terms of the GNU GPL version 3.0.
Please see qt.digia.com/product/licensing for an overview of Qt licensing.
Copyright (C) 2013 Digia Plc and/or its subsidiary(-ies).
Qt is a Digia product. See qt.digia.com for more information.
OK

LIITE F: LASKENTAGRAAFI